Der DATA BECKER Führer

# GfA BASIC

Alles auf einen Blick

# Danksagung

Anfang 2019 begann ich damit, alles an Atari-Hardware zu erwerben, was ich bereits in den 80er/90er-Jahren schon einmal besaß. So auch Bücher, die sich speziell mit dem Thema Programmierung beschäftigen. Ich stellte schnell fest, dass dieses Vorhaben heutzutage schwieriger werden könnte als der Kauf eines Atari ST. Für eine gebrauchte Ausgabe von „Das große GFA Basic 3.5 Buch“ zahlte ich mehr als den damaligen Neupreis! Mein Wunsch war aber, genau dieses Buch wieder in meiner Sammlung zu sehen. Da das Buch sehr gut erhalten war, mochte ich es auch so belassen und scannte es daher ein. Somit konnte ich nun bequem digital darin blättern, nach Begriffen suchen und Lesezeichen erstellen. Später entstand allerdings der Gedanke, dass diese digitale Version doch eigentlich der Allgemeinheit zur Verfügung gestellt werden sollte. Eine Anfrage zur Genehmigung beim Data Becker Verlag war allerdings nicht mehr möglich, da der Verlag seinen Geschäftsbetrieb zum 31.03.2014 aufgegeben hatte und es auch keine Rechtsnachfolger gibt. Somit machte ich mich auf die Suche nach dem Autor, Uwe Litzkendorf, und fand ihn auch. Mit großer Freude durfte ich feststellen, dass er meine Meinung teilte. Und mehr noch, Uwe machte den Vorschlag, dass ich es nicht nur bei diesem einem Buch belassen sollte, sondern auch weitere seiner geschriebenen Fachbücher zum Thema GFA-Basic digitalisieren dürfte. Ich war begeistert und erklärte mich gerne bereit dieses Projekt durchzuführen. So entstand aus einer einfachen E-Mail Anfrage eine wunderbare Möglichkeit den Inhalt dieser Bücher digital für die Zukunft zu erhalten und allen Interessierten kostenfrei zur Verfügung stellen zu können. Dafür möchte mich bei Uwe noch einmal sehr herzlich bedanken.

Thomas Werner
Bearbeitungsstand: 08. Mai 2020

# Neues Vorwort zur Digitalisierung

Es hat lange gedauert! Aber nun hat sich Thomas Werner die viele Arbeit gemacht, den größten Teil meiner Bücher nach, zum Teil über dreißig, Jahren in interaktiver PDF-Form zu digitalisieren und als Public Domain kostenlos online zur Verfügung zu stellen. Nachdem meine Bücher einen langen Dornröschenschlaf hinter sich haben, sollen sie und das immense darin festgeschriebene Wissen jetzt wieder zum nützlichen Leben erweckt werden :O)

Meiner Ansicht nach ist die objektorientierte Programmierung nicht für die Massenanwendung geeignet. Nach meiner Berechnung sind nur ca. 15 Prozent der Bevölkerung in der Lage, mit der OOP sinnvolle Erfolge zu erzielen. Um aber "massenfähig" zu sein, muss eine Programmiersprache mindestens 80 Prozent der Bevölkerung erreichen, damit sich Lehrer und Schüler auch außerhalb von IT-Leistungskursen über die Softwareentwicklung so verständigen können, dass der eine – vermittelbar und auch prüfbar – wenigstens "einigermaßen" versteht, was der andere meint. Andererseits "reißt unweigerlich der Faden" zwischen Ausbildern und Lernenden. Daher ist es auch kein Wunder, dass sich die prozeduralen und damit auch leicht anwendbaren Basic-Programmiersprachen, wie z.B. das sehr populäre GFA-Basic, einer gewissen Renaissance erfreuen.

Diese Tendenz werde ich natürlich als ehemals populärer Bestseller-Autor nach all meinen Möglichkeiten tatkräftig unterstützen. Ich verfüge über mehr als 4000 Seiten umfassenden, ausführlichen und auch unter den wachsamen Augen der Öffentlichkeit profund geprüften Software-Wissens.

Dieses Wissen ist auch in der heutigen Zeit absolut nicht überflüssig und veraltet, sondern bildet auch heute noch die Basis für algorithmisches Grundlagenwissen.

Aber damit nicht genug: ich habe zudem beschlossen, eine neue Programmiersprache namens "QS!X©" zu entwickeln. Sie wird sich in vielen Punkten an einfachem Standard-Basic anlehnen. Auf der weltweit überall auf allen Betriebssystemen und in jedem Standardbrowser verfügbaren – und damit 100% cross-kompatiblen – Plattform von HTML5/Canvas wird "QS!X©" als Open Source-Version (ähnlich LINUX) verfügbar sein. Nähere Informationen dazu finden Sie unter:

**http://www.litzkendorf.net/invitation_info_d.pdf**

Damit ist auch der "Klasse für die Masse"-Philosophie von Frank Ostrowski (dem GFA-Basic-Vater) Rechnung getragen. Wenn denn alles so funktioniert, wie ich es mir vorstelle, wird die weise, sanfte und erzfreundlich geduldige und bescheidene Denkart von Frank Ostrowski auch Jahre nach seinem (viel zu frühen) Ableben noch weltweit merkliche Wirkung tragen! Er bildet dann verdientermaßen – zumindest im IT-Business – die philosophische Grundlage für eine Art "Weltsprache"! Das würde ihm sicher sehr gefallen!

In treuem Gedenken an einen wirklich großen Mann, mit dem ich das Vergnügen hatte, teil- und zeitweise recht eng und vertraut zusammen zu arbeiten und dem mit "QS!X©" auch ein digitales – und verdientes – Denkmal gesetzt wird!

Uwe Litzkendorf
Hildesheim, im Mai 2020

**Bei der Nutzung und Weitergabe der vorliegenden digitalen Version ist Folgendes zu beachten:**

*Ein Weiterverkauf der digitalen Ausgabe ist nicht gestattet. Die Rechte liegen weiterhin beim Autor.*

*Eine Weitergabe dieses PDFs ist nur in unveränderter Form erlaubt*

*Ausdrucke einzelner Seiten sind für rein private Zwecke selbstverständlich gestattet.*

*Öffentliche Vorführungen – auch Auszugsweise – sind gestattet, solange diese keinen finanziellen Zwecken dienen. Ausgenommen davon sind Presseberichterstattungen.*

Auch wenn dieses Buch kostenfrei zur Verfügung gestellt wurde, hat die Erstellung einiges an privater Zeit, Geld und Arbeit gekostet. Wer dies zu schätzen weiß, darf sich gerne erkenntlich zeigen.

Weitere Informationen und eBooks finden sich unter:
http://ebook.pixas.de

*Uwe Litzkendorf*
*Michael Hösel*

*Der DATA BECKER Führer*

# GfA BASIC

ISBN 3-89011-402-4

Copyright © 1986 DATA BECKER GmbH
Merowingerstraße 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.*

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

# Inhaltsverzeichnis


| | | |
|---|---|---|
| **1** | **Der GfA-BASIC-Führer** | **9** |
| 1.1 | Zu diesem Führer | 10 |
| 1.2 | GfA-BASIC - eine Übersicht | 11 |
| **2.** | **Ein-/Ausgabebefehle** | **21** |
| 2.1 | Tastaturanweisungen | 21 |
| 2.2 | Bildschirmoperationen | 26 |
| 2.3 | Diskettenoperationen | 31 |
| 2.4 | Dateihandhabung | 40 |
| 2.5 | Druckeranweisungen | 43 |
| 2.6 | Peripherieanweisungen | 45 |
| 2.7 | Soundchip-Anweisungen | 53 |
| **3.** | **Programmstruktur** | **56** |
| 3.1 | Schleifenkonstruktionen | 56 |
| 3.2 | Kontrollbefehle | 59 |
| 3.3 | Strukturbefehle | 64 |
| **4.** | **Textoperationen** | **75** |
| **5.** | **Arithmetikbefehle** | **81** |
| 5.1 | Mathematische Operationen | 81 |
| 5.2 | Numerische Funktionen | 87 |
| **6.** | **Grafik** | **95** |
| 6.1 | Grafikdefinitionen | 95 |
| 6.2 | Grafikbefehle | 105 |
| 6.3 | Bildschirmoperationen | 114 |
| **7.** | **Datenumwandlung** | **116** |
| **8.** | **Feld-, Speicher-, Zeigerbefehle** | **123** |
| 8.1 | Feldoperationen | 123 |
| 8.2 | Speicheroperationen | 125 |
| 8.3 | Zeigeroperationen | 129 |
| **9.** | **Dialogabwicklung** | **134** |
| **10.** | **Window-Programmierung** | **137** |
| **11.** | **Menü-Programmierung** | **144** |
| **12.** | **Systemaufrufe** | **151** |
| **13.** | **Systemdaten und -adressen** | **154** |

**14. Die neuen V2.0-Befehle ..... 162**
14.1 Systemroutinen ..... 185
14.2 AES ..... 186
14.3 VDI ..... 193
14.4 GEMDOS ..... 196
14.5 BIOS ..... 199
14.6 XBIOS ..... 202

**15. Anhang ..... 208**
15.1 Direktmodus ..... 208
15.2 Der Editor ..... 209
15.3 GfA-BASIC-Fehlerliste ..... 219
15.4 Quick-Reference ..... 224
15.5 Stichwortverzeichnis ..... 235

# 1. Der GfA-BASIC-Führer

Hauptanliegen dieses Buches ist es, die über zweihundert Befehle, die vom GfA-Interpreter zur Verfügung gestellt werden, nach Schwerpunkten zu ordnen. Dies ermöglicht es, die jeweilige Befehlsbeschreibung nach problemorientierten Gesichtspunkten aufzufinden.

Das GfA-BASIC setzte mit seinem Erscheinen neue Standards: Ernsthafte Anwendungen in BASIC wurden möglich und das in einer Geschwindigkeit, die mit den üblichen Compilersprachen durchaus mithalten kann.

Hinzugekommen sind nun in der Version 2 neben etlichen BASIC-Befehlen auch komfortable Erweiterungen, die ebenfalls in diesem Führer erläutert werden. Mit diesen neuen Befehlen wird der Einsatz des zu erwartenden Compilers vorbereitet und gleichzeitig wesentlich vereinfacht.

Noch einige Erläuterungen zum Programmieren in GfA-BASIC: Grundsätzlich ist es möglich, jeden GfA-BASIC-Befehl im Direkt-Modus ("Direkt"-Feld mit der Maus anklicken oder die Esc-Taste drücken) einzugeben. Ausnahme: Schleifen- und GOTO-Anweisungen, Prozedur-Definitionen und Bedingungsabfragen. Die Rückkehr aus dem Direkt-Modus zum Editor erfolgt wahlweise mit Escape oder Eingabe von Ed, gefolgt von Return, oder durch gleichzeitiges Drücken der Tasten Control, Shift und Alternate.

GfA-BASIC bietet die Möglichkeit, eigene Editor-Routinen zu schreiben und diese aus dem Direkt-Modus aufzurufen, da der Befehl GOSUB hier auch funktioniert und somit eine Prozedur, die sich im aktuellen Programm befindet, aufgerufen werden kann. So kann unter Umständen ein Verlassen des Interpreters vermieden werden.

## 1.1 Zu diesem Führer

Dieser Führer ist so aufgebaut, daß Sie zum einen über das Stichwortverzeichnis (S. 235ff) wie auch über die Quick-Reference (S. 224ff) zum gewünschten Ziel kommen. Da die Befehle nach sog. Sachgruppen geordnet sind, bietet sich bei der Suche nach Problemlösungen die Übersicht ab Seite 11 an, um schnellstmöglich zum Ergebnis zu gelangen.

Neben klarem Aufbau liegt die Effizienz dieses Führers maßgeblich in einigen festen Vereinbarungen zur einheitlichen Darstellung, die sich konsequent durch den ganzen Text hindurchzieht:

**<Taste>** Wird bei der Befehlserläuterung auf bestimmte Tastendrücke Bezug genommen, so sind die zu betätigenden Tasten bei ihrem Namen genannt und zur besseren Kennzeichnung immer in spitze Klammern eingeschlossen.

[ ] Viele Befehle können in GfA-BASIC abgekürzt werden. Sollte eine Kurzschreibweise eines Befehls vorhanden sein, so ist diese in der Titelzeile und in der Quickreferenz dadurch gekennzeichnet, daß sie in eckige Klammern eingeschlossen ist.

.... Soll eine Folge von Anweisungen innerhalb von Befehlen verdeutlicht werden, geschieht dies anhand einer Punktlinie.

Grundsätzlich sind in der Syntaxbeschreibung alle Befehlsnamen in Großbuchstaben, alle Variablen, Parameter und Strings in üblicher Schreibweise dargestellt (z.B. OPENW Handle).

# 1.2 GfA-BASIC - eine Übersicht

## Ein-/Ausgabebefehle

### Tastaturanweisungen


FORM INPUT ..... 21
INKEY$ ..... 21
INPUT ..... 23
INPUT$ ..... 24
LINE INPUT ..... 25


### Bildschirmoperationen


POS ..... 26
PRINT ..... 26
PRINT USING ..... 27
SPC ..... 29
TAB ..... 29
WRITE ..... 30


### Diskettenoperationen


BLOAD ..... 32
BSAVE ..... 32
CHAIN ..... 33
CHDIR ..... 33
CHDRIVE ..... 34
DFREE ..... 34
DIR ..... 34
DIR$ ..... 35
EXIST ..... 36
FILES ..... 36
KILL ..... 37
LOAD ..... 37
LIST ..... 37
MKDIR ..... 38
NAME ..... 38
RMDIR ..... 39
SAVE ..... 39
PSAVE ..... 39

**Dateihandhabung**


EOF ..... 40
LOC ..... 40
LOF ..... 41
RELSEEK ..... 41
SEEK ..... 42


**Druckeranweisungen**


HARDCOPY ..... 43
LLIST ..... 43
LPOS ..... 44
LPRINT ..... 44


**Peripherieanweisungen**


CLOSE # ..... 45
FIELD # ..... 45
GET # ..... 46
INP ..... 46
INPUT # ..... 47
LINE INPUT # ..... 48
OPEN # ..... 48
OUT ..... 50
PRINT # ..... 50
PRINT # USING ..... 51
PUT # ..... 51
WRITE # ..... 52


**Soundchip-Anweisungen**


SOUND ..... 53
WAVE ..... 54


**Programmstrukturen**

**Schleifenkonstruktion**


DO..LOOP ..... 56
FOR..NEXT ..... 56
REPEAT..UNTIL ..... 57
WHILE..WEND ..... 58

## Kontrollbefehle


EXIT IF ..... 59
GOSUB ..... 59
GOTO ..... 60
IF (ELSE) ENDIF ..... 60
ON BREAK ..... 61
ON ERROR GOSUB ..... 61
ON GOSUB ..... 62
RESUME ..... 62
TRON ..... 63
TROFF ..... 63


## Strukturbefehle


CLEAR ..... 64
CLS ..... 64
CONT ..... 65
DATA ..... 65
DEFFN ..... 66
DEFLIST ..... 67
EDIT ..... 67
END ..... 68
FN ..... 68
LET ..... 69
LOCAL ..... 69
NEW ..... 70
PAUSE ..... 70
PROCEDURE ..... 70
QUIT ..... 71
READ ..... 71
REM ..... 72
RESTORE ..... 72
RETURN ..... 73
RUN ..... 73
STOP ..... 73
SYSTEM ..... 73
TIMER ..... 74

## Textoperationen


INSTR ..... 76
LEFT$ ..... 76
LEN ..... 77
LSET ..... 77
MID$ ..... 78
RIGHT$ ..... 78
RSET ..... 79
SPACE$ ..... 79
STRING$ ..... 80
UPPER$ ..... 80


## Arithmetikbefehle

## Mathematische Operationen


AND ..... 82
OR ..... 83
XOR ..... 83
NOT ..... 83
IMP ..... 83
EQV ..... 83
ADD ..... 85
DEC ..... 85
DIV ..... 85
INC ..... 85
MUL ..... 86
SUB ..... 86
SWAP ..... 86


## Numerische Funktionen


ABS ..... 87
ATN ..... 88
COS ..... 88
EXP ..... 88
FIX ..... 89
FRAC ..... 89
INT ..... 89
LOG ..... 90
LOG10 ..... 90
MAX ..... 90
MIN ..... 91
PI ..... 91

RANDOM ..... 92
RND ..... 92
SGN ..... 92
SIN ..... 93
SQR ..... 93
TAN ..... 93
TRUNC ..... 94

## Grafik

### Grafikdefinition

COLOR ..... 95
DEFFIL ..... 96
DEFLINE ..... 98
DEFMARK ..... 99
DEFMOUSE ..... 99
DEFTEXT ..... 101
GRAPHMODE ..... 103
SETCOLOR ..... 104

### Grafikbefehle

BOX ..... 105
PBOX ..... 105
CIRCLE ..... 106
PCIRCLE ..... 106
DRAW ..... 107
ELLIPSE ..... 107
PELLIPSE ..... 107
FILL ..... 108
LINE ..... 108
PLOT ..... 108
POINT ..... 109
POLYLINE ..... 109
POLYFILL ..... 110
POLYMARK ..... 110
RBOX ..... 111
PRBOX ..... 111
SPRITE ..... 111
TEXT ..... 112

### Bildschirmoperationen

GET ..... 114
PUT ..... 114

### Datenumwandlung

ASC ..... 116
BIN$ ..... 117
CHR$ ..... 117
CVI ..... 118
CVL ..... 118
CVS ..... 118
CVF ..... 118
CVD ..... 118
HEX$ ..... 119
MKI$ ..... 119
MKL$ ..... 119
MKS$ ..... 119
MKF$ ..... 119
MKD$ ..... 119
OCT$ ..... 120
STR$ ..... 121
VAL ..... 121
VAL? ..... 122

## Feld-, Speicher und Zeigerbefehle

### Feldoperationen

ARRAYFILL ..... 123
DIM ..... 123
DIM ? ..... 124
ERASE ..... 124

### Speicheroperationen

DPEEK ..... 125
LPEEK ..... 125
PEEK ..... 125
DPOKE ..... 126
LPOKE ..... 126
POKE ..... 126

SDPOKE ..... 127
SLPOKE ..... 127
SPOKE ..... 127
C: ..... 127
CALL ..... 128

**Zeigeroperationen**

ARRPTR ..... 133
VARPTR ..... 133

**Dialogabwicklung**

ALERT ..... 134
FILESELECT ..... 135
MOUSE ..... 136
MOUSEX ..... 136
MOUSEY ..... 136
MOUSEK ..... 136

**Window-Programmierung**

CLEARW ..... 137
CLOSEW ..... 137
FULLLW ..... 137
INFOW ..... 138
OPENW ..... 138
TITLEW ..... 141
WINDTAB ..... 141

**Menü-Programmierung**

MENU(Index) ..... 144
MENU ..... 146
MENU KILL ..... 148
MENU OFF ..... 148
ON MENU ..... 148
ON MENU GOSUB ..... 149
ON MENU IBOX GOSUB ..... 149
ON MENU OBOX GOSUB ..... 1490
ON MENU KEY GOSUB ..... 150
ON MENU MESSAGE GOSUB ..... 150

## Systemaufrufe

GEMSYS ..... 151
VDISYS ..... 152
BIOS ..... 153
XBIOS ..... 153
GEMDOS ..... 153

## Systemdaten und -adressen

ADDRIN ..... 154
ADDROUT ..... 154
CONTRL ..... 154
DATE$ ..... 155
ERR ..... 156
ERROR ..... 156
FATAL ..... 156
FRE ..... 157
GB ..... 157
GCONTRL ..... 158
GINTIN ..... 158
GINTOUT ..... 159
INTIN ..... 159
INTOUT ..... 159
PTSIN ..... 160
PTSOUT ..... 160
SETTIME ..... 160
TIME$ ..... 161

## Die neuen V2.0-Befehle

! ..... 165
* ..... 166
BASEPAGE ..... 166
BITBLT ..... 167
BGET ..... 168
BMOVE ..... 169
BPUT ..... 169
CLR ..... 169
CRSCOL ..... 170
CRSLIN ..... 170
DEFNUM ..... 170
EVEN ..... 171
EXEC ..... 171
FALSE ..... 173

FORM INPUT AS ..... 173
HIDEM ..... 173
HIMEM ..... 174
INP? ..... 174
MID$() ..... 175
MONITOR ..... 175
ODD ..... 176
ON MENU BUTTON GOSUB ..... 176
OPTION ..... 178
OPTION BASE ..... 178
OUT? ..... 179
RESERVE ..... 179
SGET ..... 180
SHOWM ..... 181
SPUT ..... 181
TRUE ..... 181
TYPE ..... 182
VDIBASE ..... 182
VOID ..... 184
VSYNC ..... 182

**Systemroutinen**

**AES**

GRAF_HANDLE ..... 186
APPL_READ ..... 186
APPL_WRITE ..... 187
RSRC_LOAD ..... 187
RSRC_FREE ..... 187
RSRC_GADDR ..... 187
FORM_DO ..... 188
FORM_CENTER ..... 188
OBJC_DRAW ..... 188
OBJC_OFFSET ..... 188
MENU_TEXT ..... 189
GRAF_RUBBERBOX ..... 190
GRAF_DRAGBOX ..... 190
GRAF_MOVEBOX ..... 191
GRAF_GROWBOX ..... 191
GRAF_SHRINKBOX ..... 192
GRAF_MKSTAT ..... 192

## VDI


Set Grafik Text Special Effects ..... 193
Set Character Cell High ..... 193
Set Clipping Rectangle ..... 194
Set Fill Perimeter Visibility ..... 194
Inquire Color Representation ..... 195
Current Fill Area Attributs ..... 195


## GEMDOS


Set DRV ..... 196
Current Disk ..... 197
Set Disk Transfer Adress ..... 197
Get DTA ..... 197
Malloc ..... 198
Mfree ..... 198
Sfirst ..... 198


## BIOS


GETMPB ..... 199
RWABS ..... 199
GETMPB ..... 200
MEDIACH ..... 200
DRVMAP ..... 201
KBSHIFT ..... 201


## XBIOS


PHYSBASE ..... 202
LOGBASE ..... 202
GETREZ ..... 202
SETSCREEN ..... 203
SETCOLOR ..... 203
FLOPRD ..... 204
FLOPWR ..... 205
FLOPFMT ..... 205
MIDIWS ..... 206
FLOPVER ..... 206
SETPRT ..... 206
KBRATE ..... 207
PUNTAES ..... 207

# 2. Ein-/Ausgabebefehle

## 2.1 Tastaturanweisungen

### FORM INPUT — Eingabe formatiert

**FORM INPUT Anzahl,V_name$**

Dieser Befehl eignet sich für Stringeingaben, die auf eine bestimmte Länge begrenzt werden sollen. Anders als bei *INPUT$* wird die Eingabe bei Erreichen dieser Zeichenanzahl nicht automatisch abgebrochen, sondern muß auf jeden Fall mit <RETURN> quittiert werden. Wird die letzte Eingabeposition erreicht, ertönt ein akustisches Signal. Die maximale Eingabelänge ist auf 255 Zeichen beschränkt. Die Eingabe kann mit <DELETE>, <BACKSPACE> und den vier Pfeiltasten korrigiert werden. Die Eingabe von Sonderzeichen erfolgt in der gleichen Weise wie unter *INPUT* beschrieben.

### INKEY$ — Einzelzeichen von Tastatur abfragen

**Zeichen$=INKEY$**
(ordnet das Zeichen der Variablen *Zeichen$* zu)

**IF/WHILE/UNTIL LEN(INKEY$)**
(Bedingung bei Tastendruck, wenn *INKEY$* > "")

**IF INKEY$="Z"**
(Bedingung, wenn "Z" gedrückt wird)

Ohne das Programm zu unterbrechen, wird festgestellt, ob entweder eine ASCII-Wert liefernde Taste oder eine Sondertaste (Pfeil-, Funktionstasten usw.) betätigt wurde. Es kann jeweils nur eine einzelne Taste ermittelt werden. Wird bei Ausführung des Befehls keine Taste gedrückt, wird ein Nullstring ("") geliefert. Bei ASCII-Tasten wird das ent-

sprechende Zeichen zurückgegeben. Sobald bei Ausführung eine Sondertaste gedrückt wird, liefert *INKEY$* einen Zwei-Zeichen-String, dessen erstes Zeichen immer ein Nullbyte ist. Das zweite Zeichen beinhaltet dann den Pseudo-ASCII der jeweiligen Taste. Wie aus den Syntaxvarianten zu ersehen ist, muß das ermittelte Zeichen nicht in jedem Fall einer Stringvariablen zugeordnet werden. *INKEY$* kann auch direkt in Bedingungsabfragen eingebunden werden.

```
Do
 Zeichen$=Inkey$
 Zeich.1=Asc(Left$(Zeichen$))
 Zeich.2=Asc(Right$(Zeichen$))
 Zeich.3=Zeich.1*256+Zeich.2
 Print At(10,10);Spc(69)
 Print At(10,10);
 Print Using "! ### ### #####", Zeichen$,
Zeich.1, Zeich.2,Zeich.3
 ' liefert Zeichen, ASCII-Code, Scan-Code und
 ' Word-Wert der gedrückten Taste
 Repeat
  ' leert den Tastaturpuffer
  Until Inkey$=""
  Pause 10
 Loop
```

**INPUT V_name**
(ohne Kommentar)

**INPUT "Text"[;,]V_name**
(mit Kommentar)

**INPUT V_name1$,V_name2 ...**
(mehrere Variablen)

**INPUT "Text";V_name1,V_name2...**
(mit Kommentar)

Das Programm wird unterbrochen und der Anwender zur Dateneingabe aufgefordert. Diese Daten werden Variablen zugeordnet.

**"Text"** ist eine beliebige Zeichenkette (immer in Anführungsstrichen), die optional verwendet werden kann. Sie enthält einen Kommentar zur Eingabe. *V_name* steht hier für den Namen einer beliebigen Variablen, der die jeweilige Eingabe zugeordnet wird. Das Trennzeichen zwischen *"Text"* und *V_name* bestimmt die Position des Cursors im Anschluß an *"Text"*:

**;** Fragezeichen und Leerzeichen hinter dem Text

**,** Eingabe beginnt direkt rechts neben dem Text

Die Eingabe muß mit <RETURN> bestätigt werden. Passen die eingegebenen Zeichen nicht zum Datentyp der angegebenen Variablen, ertönt ein akustisches Signal und die Eingabe muß wiederholt werden.

Ein Komma in der Dateneingabe bewirkt nur Berücksichtigung des Teils vor dem Komma bzw. eine Datentrennung, falls auf den INPUT-Befehl mehrere, durch Kommata getrennte Variablen folgen. In diesem Falle können Daten auch durch <RETURN> getrennt werden. Sollen auch

Kommata in die Eingabe mit einbezogen werden, sind diese in Anführungsstriche zu setzen und gesondert einer Variablen zuzuordnen.

Bei Stringeingaben ist die Länge auf 255 Zeichen beschränkt. Die Eingabe kann mit <DELETE>, <BACKSPACE> und den vier Pfeiltasten korrigiert werden.

Die Eingabe von Sonderzeichen über Tastatur kann folgendermaßen vorgenommen werden:

- <ALTERNATE>-Taste zusammen mit anderer Taste

- <CONTROL> und <S>-Taste gleichzeitig, gefolgt von einer anderen Taste

- <CONTROL> und <A>-Taste gleichzeitig, gefolgt vom jeweiligen ASCII-Code des Zeichens. Zeichen mit einem ASCII-Wert zwischen 0 und 32 müssen nach Eingabe des Wertes mit <RETURN> quittiert werden.

## INPUT$ — Eingabe einer Zeichenkette

**A$=INPUT$(Anzahl)**
(Eingabe über Tastatur)

**A$=INPUT$(Anzahl,#Kanal)**
(Einlesen aus Datenkanal)

Das Programm wird unterbrochen und die Eingabe einer vorgegebenen Anzahl von Zeichen über die Tastatur oder aus einer Datei erwartet. Mit *Anzahl* kann eine maximale Stringlänge angegeben werden, nach deren Erreichen die Eingabe automatisch abgebrochen wird. Quittieren mit <RETURN> ist nur notwendig, wenn der eingegebene String die vorgegebene Zeichenanzahl nicht erreicht. Die Edition der Eingabe erfolgt wie unter *INPUT* beschrieben. Der optionale Zusatz *Kanal* bewirkt, daß die angegebene Zeichenanzahl aus der Datei mit der Kanalnummer *Kanal* gelesen wird. Der String kann maximal 32767 Zeichen aufnehmen.

## LINE INPUT [LI INPUT] Zeichenketteneingabe

**LINE INPUT "Text"[;,]V_name$**
(mit Text)

**LINE INPUT V_name$**
(ohne Text)

**LINE INPUT V_name1$,V_name2$ ...**
(mehr Variablen)

**LINE INPUT "Text";V_name1$,V_name2$...**
(beides)

Ermöglicht die Eingabe einer Zeichenkette während des Programmlaufes. Während beim INPUT-Befehl Kommata eine Trennfunktion haben, werden hier Kommata in den einzugebenden Text mit aufgenommen. Bei Mehrfacheingaben können die einzelnen Strings nur durch Betätigung der <RETURN>-Taste voneinander getrennt werden. Die Strings werden der jeweiligen Variablen zugeordnet. Der optionale Zusatz *"Text"* kann einen beliebigen Eingabekommentar enthalten. Die Editionsmöglichkeiten entsprechen den unter *INPUT* beschriebenen.

## 2.2 Bildschirmoperationen

### POS — Cursorspalte ermitteln

**POS(Dummy)**

Es kann die Bildschirmspalte ermittelt werden, in welcher sich der Cursor aktuell befindet. Da eine Ausgabezeile bis zu 256 Zeichen lang sein kann, wird durch *POS* ein Wert von 0 bis 255 zurückgegeben. Diese Position muß nicht immer mit der realen Cursorposition identisch sein. *Dummy* kann ein beliebiger Wert sein. Er ist ohne Bedeutung, muß jedoch angegeben werden (dummy = engl. Attrappe).

### PRINT [? oder P] — Daten ausgeben

**PRINT**
**PRINT "Text"**
**PRINT,"Text"[;,']"Text"[;,']V_name[;,']V_name$[;,']...**
**PRINT AT(S,Z)[;,']"Text"[;,']Ausdruck[;,']V_name**

Die vorstehenden Syntaxmuster zeigen nur eine begrenzte Auswahl der Anwendungsmöglicheiten dieses Befehls. *PRINT* ohne Angabe von Daten, Texten oder Formatzeichen bewirkt, daß eine Leerzeile ausgegeben wird, die mit einem CARRIAGE-RETURN und LINE FEED abgeschlossen wird. Der Cursor springt zum Anfang der nächsten Zeile. *PRINT* nur von einem Wert, einer Variablen oder einem Textausdruck gefolgt, gibt den Wert, den Inhalt der Variablen oder den Text aus und schließt die Zeile mit CR und LF ab. Mit Formatzeichen kann die Ausgabe modifiziert werden:

; bewirkt eine Unterdrückung des CR und LF, die folgende Ausgabe schließt sich an das letzte Zeichen der letzten Ausgabe an. Ist die ausgegebene Zeile länger als 80 Zeichen, tritt ein Zeilenüberlauf ein. Die Zeile wird am Anfang der nächsten Bildschirmzeile fortgesetzt. Soll am Seitenende das Zeilenscrolling unterdrückt werden, ist auf jeden Fall vor Erreichen der letzten Bildschirmposition ein Semikolon zu setzen.

, Wird als Trennzeichen ein Komma verwendet, wird das nächste auszugebende Zeichen an den nächsten von fünf Tabulatorpunkten gesetzt (1,17,33,49,65).

' Die Funktion des Apostrophs ist weitgehend mit der des Semikolons identisch. Zusätzlich wird jedoch an Stelle des Apostrophs ein Leerzeichen ausgegeben.

Mit dem optionalen Zusatz *AT(S,Z)* kann der Cursor an einer bestimmten Bildschirmposition (Spalte,Zeile) plaziert werden. Der Startindex ist in beiden Dimensionen 1 (*AT(0,0)* ist identisch mit *AT(80,25)*). Innerhalb von GEM-Fenstern kann der Ausgabemodus mit *DEFTEXT* und *GRAPHMODE* eingestellt werden.

## PRINT USING [P USING] Formatiert ausgeben

**PRINT USING "format", Ausdruck$, V_name$, Ausdruck, V_name**

Dem Befehl wird ein String übergeben, der die gewünschten Formatangaben enthält. Entsprechend dieser Angaben werden dann die Texte und Werte zur Ausgabe vorbereitet und anschließend ausgegeben. Statt eines Textausdruckes kann auch eine Stringvariable übergeben werden, die die Formatangaben enthält.

Folgende Formatsymbole sind möglich:

| | |
|---|---|
| # | dieses Zeichen steht im Formatstring stellvertretend für eine einzelne Ziffer. |
| . | gibt an, an welcher Stelle der formatierten Zahl ein Dezimalpunkt vorgesehen werden soll. |
| + | bewirkt, daß auch ein positives Vorzeichen mit ausgegeben wird. |
| - | falls der auszugebende Wert negativ ist, erscheint an dieser Stelle das Minuszeichen. |
| * | dieses Zeichen vor dem Dezimalpunkt bewirkt, daß alle Vorkommastellen, die nicht von dem auszugebenden Wert belegt werden, mit "*" ausgegeben werden. Hinter dem Dezimalpunkt verwendet, werden soviele "*" ausgegeben, wie angegeben wurden und die auszugebende Zahl wird gerundet (< .5 = abrunden / > .5 = aufrunden). |
| $ | zur Darstellung hexadezimaler Werte kann hiermit die Voranstellung eines "$" vor dem Hexadezimal-String bewirkt werden. |
| , | hiermit können zur Tausendertrennung Kommata in die auszugebende Zahl eingefügt werden. |
| ^^^^ | durch Angabe dieser Zeichen kann die Anzahl der Platzhalter zur Darstellung der auszugebenden Zahl im Exponentialformat bestimmt werden. |
| ! | das erste Zeichen des übergebenen Strings wird ausgegeben. |
| & | der gesamte übergebene String wird ausgegeben. |

\....\ vom Stringanfang ausgehend werden soviele Zeichen eines Strings ausgegeben, wie Platzhalter dieser Art (einschl. <BACKSLASH>) vorgegeben wurden.

_ (Tiefstrich) bewirkt die Ausgabe des nächsten Zeichens eines Strings.

## SPC — Leerzeichen ausgeben

**SPC(Anzahl)**

Dieser Befehl gibt eine vorgegebene Anzahl von Leerzeichen aus und ist nur im Zusammenhang mit dem PRINT-Befehl verwendbar. Er läßt sich also nicht in Stringkonstrukte integrieren. Innerhalb eines PRINT-Ausdrucks kann er an jeder beliebigen Position eingesetzt werden.

## TAB — Tabulator setzen

**TAB(Position)**

Es kann eine Tabulatorposition bestimmt werden, an der die nachfolgende PRINT-Anweisung ausgeführt wird. *Position* kann im Bereich zwischen 0 und 255 liegen. Größere Werte werden mit Mod 256 auf diesen Bereich zurückgerechnet. Befindet sich der Cursor in einer Zeile hinter der zuletzt angegebenen Tabulatorposition und liegt der Wert einer sich anschließenden Tabulator-Anweisung zwischen 256 und der aktuellen Cursorposition, wird dieser Tabulator in derselben Zeile ausgeführt. Ist ein Tabulatorwert kleiner als die aktuelle Cursorposition, wird die Anweisung erst in der nächsten Zeile wirksam. Die Tabulator-Anweisung ist nur in Verbindung mit dem PRINT-Befehl ausführbar. Es ist nicht möglich, sie in einen Stringausdruck einzubinden.

**WRITE**
**WRITE "Text"**
**WRITE "Text","Text",V_name,V_name$,Ausdruck;**

*WRITE* kann zur Text-, bzw. Datenausgabe auf dem Bildschirm verwendet werden und ist dem PRINT-Befehl sehr ähnlich, hat jedoch einen anderen syntaktischen Aufbau. Die Anführungszeichen eines übergebenen Strings (bzw. Ausdruck, Variable) sowie Kommata, welche die einzelnen Listenausdrücke voneinander trennen, werden ebenfalls ausgegeben. Zur Unterdrückung des Zeilenscrolling kann als Listenabschluß ein Semikolon verwendet werden. *WRITE* ohne weitere Zusätze bewirkt wie *PRINT* die Ausgabe einer Leerzeile. Weitere Verwendungsmöglichkeiten unter *WRITE#*.

## 2.3 Diskettenoperationen

Bei allen Diskettenoperationen, denen ein Dateiname zu übergeben ist, besteht die Möglichkeit, mit diesem einen Suchpfad zu definieren, über den der Dateizugriff ausgeführt werden soll:

- Laufwerkbezeichnung: Buchstabe mit nachfolgendem Doppelpunkt, wie A: für das erste Diskettenlaufwerk, B: für das zweite...

- Fragezeichen innerhalb des Dateinamens sind Platzhalter. Es werden alle Dateien angesprochen, die bis auf das/die Fragezeichen mit dem angegebenen Dateinamen übereinstimmen.

- Sternchen im Dateinamen sind Platzhalter für einen ganzen Bereich (vor oder nach dem Trennpunkt).

Beispiele:

**B:\UTILITY*.***
sucht auf Disk B nur im Ordner "Utility" nach allen auftretenden Dateien.

**B:\UTILITY\OUTPUTS*.WAS**
sucht nur im Unterordner "Outputs" nach allen Dateien mit der Extension.

***.WAS** sucht auf der aktuellen Disk in der Hauptdirectory nach allen auftretenden Dateien mit der Extension ".WAS".

**\DR????** sucht auf der aktuellen Diskette nach allen Dateien mit einem sechs Buchstaben langen Namen, der mit "DR" beginnt.

**A:\DR????.WAS**
sucht auf Disk A nach allen Dateien mit einem sechs Buchstaben langen Namen, der mit "DR" beginnt und die Extension ".WAS" trägt.

**\D*.*** sucht auf der aktuellen Diskette nach allen Dateien, die mit dem Buchstaben "D" beginnen.

## BLOAD [BL] Bereich laden

**BLOAD "Dateiname",Start**

Es werden Datenblöcke, die zuvor mit *BSAVE* auf Diskette gespeichert wurden, in den Arbeitsspeicher zurückgeladen. *Dateiname* gibt den Namen der zu ladenden Datei an (Pfadstruktur siehe Kapitelanfang). Es ist ein Parameter zu übergeben, anhand dessen bestimmt wird, an welche Speicherstelle der Block geladen werden soll. Eine Blocklänge braucht nicht angegeben zu werden.

## BSAVE [BS] Bereich speichern

**BSAVE "Dateiname",Start,Anzahl**

*BSAVE* bietet die Möglichkeit, eine vorgegebene Anzahl von RAM-Bytes als Block auf Diskette zu speichern. Es ist der Name der Datei anzugeben, in welcher der Block gespeichert werden soll. Nach einem Komma folgt die RAM-Adresse des ersten Bytes des Blockes (*Start*) und abschließend wieder nach einem Komma die *Anzahl* der Bytes, die von dieser Adresse ab gelesen und gespeichert werden soll.

## CHAIN [CH] Programm laden (Autostart)

**CHAIN "Programmname"**

*CHAIN* lädt ein BASIC-Programm von Diskette und startet es selbsttätig. Es erlaubt die Verkettung von Programmen je nach Bedarf. Das Laden eines Programmes bewirkt, daß der BASIC-Arbeitsspeicher samt Inhalt (aufrufendes Programm) und Variablenbelegung gelöscht wird. Besitzt *Programmname* keine Extension, fügt der Interpreter die Extension *".BAS"* an. *Programmname* ist gemäß hierarchischem Filesystem möglich. (siehe Kapitelanfang)

## CHDIR [CHD] Ordner wechseln

**CHDIR "Ordnername"**

*CHDIR* ermöglicht es, einen vorhandenen Ordner als aktuellen Ordner für Diskettenzugriffe zu bestimmen. Wird der neue *Ordnername* allein angegeben, erfolgt jeder folgende Diskettenzugriff innerhalb dieses Ordners, bei Angabe eines Suchpfades ("\Ordner\Unterordner...") gemäß dem hierarchischen Filesystem (siehe "Diskettenoperationen") kann eine ganze Ordner-Unterordner-Folge bestimmt werden. Möchte man eine tiefere hierarchische Ebene (abwärts) ansprechen, kann das Voranstellen eines BACKSLASH ("\") unterbleiben. Soll jedoch der Ordner in Richtung höherer Ebene (aufwärts) verlassen werden, muß der BACKSLASH vor dem Suchpfad angegeben werden. Besteht der Ordnername nur aus dem BACKSLASH ("\"), wird danach nicht auf einen Ordner, sondern auf das Hauptdirectory zugegriffen. *CHDIR* greift grundsätzlich nur auf das aktuelle Laufwerk zu, ein Laufwerkswechsel ist nicht möglich (siehe dazu *CHDRIVE*). Die Anführungszeichen können bei "Ordner" vernachlässigt werden.

## CHDRIVE [CHDR] Laufwerk (aktuell) bestimmen

**CHDRIVE Laufwerk**

Falls bei einem nachfolgenden Befehl mit Diskettenzugriff keine Laufwerks-Bezeichnung angegeben wird, kann mit *CHDRIVE* anhand der Kennung *Laufwerk* (1=Laufw.A; 2=Laufw.B...15=Laufw.O) bestimmt werden, welches Laufwerk als das aktuelle gelten soll.

## DFREE Freien Disketten-Speicherplatz ausgeben

**DFREE(Laufwerk)**

Ermittelt, wieviel freier Speicherplatz auf einer bestimmten Diskette zur Verfügung steht. Der Wert wird in Byte ausgegeben, *Laufwerk* ist die Nummer des abgefragten Laufwerks:

**DFREE(0)**
(Aktuelles Laufwerk)

**DFREE(1..15)**
(Laufwerk A ... O)

## DIR Directory ausgeben

**DIR**
**DIR "Dateienbezeichnung"**
**DIR "Dateienbezeichnung" TO "Dateiname"**
**DIR "Dateienbezeichnung" TO "LST:"**

*DIR* erstellt das Inhaltsverzeichnis einer Diskettenstation oder eines Ordners. Diese Dateiauflistung kann auf dem Bildschirm ausgegeben werden, in eine Diskettendatei *Dateiname* oder in eine virtuelle Datei (z.B. *"CON:"* für Konsole) mit Hilfe des optionalen Zusatzes *TO* geschrieben werden. Die Ausgabe auf dem Drucker erfolgt mit dem Zusatz *"LST:"* (siehe auch unter *OPEN*). Zur Pfadangabe

können sämtliche am Kapitelanfang beschriebenen Spezifikationen verwendet werden.

Entfällt die Laufwerkbezeichnung, werden die Dateien des aktuellen Diskettenlaufwerks gelistet.

**Dir "*.*"**
listet alle Dateien, die sich in der Directory der aktuellen Disk befinden, auf dem Bildschirm.

**Dir "B:\Ordner" To "VID:"**
listet alle Dateien, die sich im Ordner *"Ordner"* auf Diskstation B befinden, in die virtuelle "VID:"-Datei (d.h., auf dem Bildschirm, ohne Ausführung von Steuerzeichen).

## DIR$ — Ordnername (aktuell) ausgeben

**DIR$(Laufwerk)**

*DIR$* ermittelt den Namen des zuletzt geöffneten Ordners und gibt ihn gegebenenfalls aus. *Laufwerk* steht für die Laufwerk-Kennung der Diskstation, auf der nachgesehen werden soll:

| | | |
|---|---|---|
| Laufwerk = | 0 | aktuelles Laufwerk |
| Laufwerk = | 1..15 | Laufwerk A..O |

Der aktuelle Ordnername kann (z.B. mit *PRINT DIR$(0)*) ausgegeben oder einer Stringvariablen übergeben werden (z.B. *A$=DIR$(0)*). Ist kein Ordner geöffnet, wird ein Leerstring zurückgegeben.

## EXIST Datei suchen

**EXIST(Dateiname)**

*EXIST* ermittelt, ob auf der angegebenen Diskette eine bestimmte Datei vorhanden ist. Es wird entweder eine 0 (nicht vorhanden) oder -1 (vorhanden) zurückgegeben. Dieser Wert kann in eine Variable überführt werden (*A=EXIST(Dateiname)*) oder direkt in einer Abfrage verwendet werden (*IF EXIST (Dateiname)<>0*). Zur hierarchischen Struktur der Pfadangabe in *Dateiname* siehe Kapitelanfang.

## FILES [FILE] Directory (erweitert) ausgeben

**FILES**
(alle Dateien)

**FILES "Dateienbezeichnung"**
(Bildschirm)

**FILES "Dateienbezeichnung" TO "Dateiname"**
(Disklisting)

**FILES "Dateienbezeichnung" TO "LST:"**
(Druckerlisting)

*FILES* erstellt das Inhaltsverzeichnis einer Diskettenstation oder eines Ordners, zusammen mit den Angaben über Dateilänge (in Byte), Erstellungs-Uhrzeit und -Datum. Diese Dateiauflistung kann auf dem Bildschirm ausgegeben werden (*FILES*), in eine Diskettendatei *Dateiname* oder in eine virtuelle Datei (z.B. *"CON:"* für Konsole) mit Hilfe des optionalen Zusatzes *TO* geschrieben werden. Die Ausgabe auf dem Drucker erfolgt mit dem Zusatz *"LST:"*. Zur Pfadangabe in *Dateiname* können sämtliche am Kapitelanfang beschriebenen Spezifikationen verwendet werden. Entfällt die Laufwerksbezeichnung, wird auf das aktuelle Diskettenlaufwerk zugegriffen.

## KILL [K] Datei löschen

**KILL "Dateiname"**

Es wird eine bestimmte Datei von der Diskette gelöscht. *Dateiname* ist der Name der zu löschenden Datei (Pfadstruktur siehe Kapitelanfang). Die Verwendung von Anführungszeichen bei *Dateiname* ist optional.

## LOAD [LOA] Programm laden

**LOAD "Programmname"**

Es kann ein beliebiges BASIC-Programm in den Arbeitsspeicher geladen werden. *Programmname* ist der Name des zu ladenden Programmes (Pfadstruktur siehe Kapitelanfang). Wird dem Programmnamen keine Extension angefügt, wird vom Interpreter selbsttätig die Extension *".BAS"* verwendet. Dieser Befehl ist grundsätzlich identisch mit der LOAD-Funktion im Editormenü.

## LIST [LIS] Programm listen/speichern (ASCII)

**LIST**
(Listing auf Bildschirm)

**LIST "Dateiname"**
(Listing in eine Datei)

*LIST* ermöglicht das Auflisten des Programmtextes auf dem Monitor oder als ASCII-Text in eine Diskettendatei. Wird der Befehlsname ohne weitere Zusätze verwendet, wird das Programmlisting auf dem Monitor ausgegeben. Dieser Vorgang kann durch die übliche Abbruchfunktion CONTROL / SHIFT / ALTERNATE abgebrochen werden. Das Listing wird als sogenanntes ASCII-File auf Diskette abgespeichert, wenn dem Befehl der Name der Datei übergeben wird, in welcher das Listing abgelegt werden soll (Pfadstruktur siehe Kapitelanfang). Dieser Befehl ist abso-

lut identisch mit der Editorfunktion *SAVE,A*. Es können also hiermit abgespeicherte Programme jederzeit mit der Editorfunktion *MERGE* in ein anderes, im Arbeitsspeicher befindliches Programm eingebunden werden. Die Anführungsstriche um *Dateiname* können vernachläsigt werden, da sie der Interpreter selbsttätig setzt. Wird zum Dateinamen keine Extension angegeben, wird diese vom Interpreter mit *".LST"* eingesetzt.

## MKDIR [MK] Ordner erzeugen

**MKDIR "Ordnername"**

*MKDIR* erzeugt einen neuen Ordner in der aktuellen Directory. *Ordnername* beinhaltet den Ordnernamen und gegebenenfalls die Pfadbezeichnung zu der Ebene, auf welcher der neue Ordner angelegt werden soll (Pfadstruktur siehe Kapitelanfang). Wird nur der reine Dateiname verwendet, wird der Ordner auf der aktuellen Ebene erzeugt (siehe *CHDIR*).

## NAME [NA] Datei umbenennen

**NAME "Name_alt" AS "Name_neu"**

Der Name einer bestehenden Datei kann durch einen neuen Namen ersetzt werden. Beide Namen können entweder als Stringausdruck, als Stringvariable oder als Kombination von beidem übergeben werden. Dieses ist übrigens bei den meisten Befehlen der Fall, welche die Angabe eines Dateinamens erwarten (Pfadstruktur siehe Kapitelanfang). Die Laufwerks-Spezifikationen müssen bei beiden Namen identisch sein. Soll der Befehl auf der aktuellen Directory ausgeführt werden, kann diese entfallen. Innerhalb einer Station kann auch ohne weiteres eine im Hauptdirectory verzeichnete Datei durch eine entsprechende Angabe im neuen Namen in einen Ordner verlegt werden, falls der angegebene Ordner vorhanden ist.

## RMDIR [RM] Ordner löschen

**RMDIR "Ordnername"**

RMDIR löscht den angegebenen Ordner. *Ordnername* enthält den Namen des zu löschenden Ordners (Pfadstruktur siehe Kapitelanfang). Bevor ein Ordner gelöscht werden kann, sind alle evtl. sich darin befindenden Dateien zu löschen oder zu verlegen. Der Ordner muß absolut leer sein.

## SAVE [SA] Programm speichern (codiert)

**SAVE "Programmname"**

Der Befehl speichert das im Arbeitsspeicher befindliche Programm unter dem angegebenen Namen codiert auf Diskette und ist identisch mit der Menüfunktion *SAVE*. *Programmname* enthält den Namen der Datei, in welcher der Programmcode abgelegt werden soll (Pfadstruktur siehe Kapitelanfang). Ist keine Extension angegeben, wird diese vom Interpreter selbsttätig mit *.BAS* eingesetzt. Sollte sich bereits ein Programm mit demselben Namen auf der angesprochenen Diskette befinden, wird dieses durch das neu zu speichernde Programm überschrieben.

## PSAVE [PS] Programm speichern (listgeschützt)

**PSAVE "Programmname"**

Speichert das im Arbeitsspeicher befindliche Programm unter dem angegebenen Namen codiert und listgeschützt auf Diskette. Für diesen Befehl gilt die gleiche Ausführung wie zu *SAVE*. Die Unterschiede zu *SAVE* sind, daß Programme direkt nach dem Laden gestartet werden und nicht mehr gelistet werden können.

## 2.4 Dateihandhabung

### EOF — Dateiende ermitteln

**EOF (#Kanalnummer)**

Es kann bei Diskettendateien das letzte Byte der Datei mit der angegebenen Kanalnummer ermittelt werden. Um z.B. die Fehlermeldung "Fileende erreicht" zu vermeiden, ist dieser Befehl einsetzbar. *Kanalnummer* enthält den Identifikator (0-99) der Datei, deren letztes Byte festgestellt werden soll. Befindet sich der Filepointer auf dem letzten Byte der Datei, wird der Wert -1 geliefert. Ist das Fileende noch nicht erreicht, erhält man den Wert 0. Dieser Wert kann entweder einer Variablen übergeben werden (*A=EOF(#1)*) oder in Abfragen direkt verwendet werden.

### LOC — Filepointerposition

**LOC (#Kanalnummer)**

Es kann der Abstand des Filepointers zum Dateianfang ermittelt werden. Der zu jeder geöffneten Datei gehörige Schreib- und Lesezeiger zeigt auf die Position in der Datei, die aktuell bearbeitet wird. Der in diesem Zeiger enthaltene Wert kann mit diesem Befehl ermittelt werden. Es wird hier immer relativ zum Dateianfang gezählt. Der Filepointer befindet sich jeweils auf dem Byte, dessen Nummer durch *LOC* geliefert wird. *Kanalnummer* enthält den Identifikator (0-99) der Datei, die angesprochen werden soll.

## LOF — Dateilänge ermitteln

**LOF (#Kanalnummer)**

LOF liefert die Länge einer Datei in Byte. *Kanalnummer* enthält den Identifikator (0-99) der Datei, deren Länge ermittelt werden soll.

## RELSEEK [REL] — Filepointer verschieben

**RELSEEK #Kanalnummer,Byteanzahl**
(Richtung Dateiende)

**RELSEEK #Kanalnummer,-Byteanzahl**
(Richtung Dateianfang)

*RELSEEK* verschiebt den Filepointer der angegebenen Datei um eine bestimmte Anzahl von Bytes entweder in Richtung Dateiende oder Dateianfang. *Kanalnummer* enthält den Identifikator (0-99) der Datei, deren Filepointer verschoben werden soll. Mit *Byteanzahl* wird die Anzahl der Bytes angegeben, um die der Filepointer verschoben werden soll. Die Angabe eines negativen Wertes bewirkt, daß von der aktuellen Position aus in Richtung Dateianfang gezählt wird. Bei positiven Werten wird in Richtung Dateiende gezählt. Es ist darauf zu achten, daß der Zeiger weder vor den Dateianfang noch hinter das Dateiende gesetzt wird.

## SEEK [SEE] Filepointer setzen

**SEEK #Kanalnummer,Bytenummer**
(ab Dateianfang)

**SEEK #Kanalnummer,-Bytenummer**
(ab Dateiende)

Es kann der Filepointer einer Datei auf ein beliebiges Byte dieser Datei gesetzt werden. *Kanalnummer* enthält den Identifikator (0-99) der Datei, deren Filepointer gesetzt werden soll. Mit *Bytenummer* wird das Byte angegeben, auf welches der Filepointer zeigen soll. Dieser Wert ist kleiner oder gleich der Dateilänge anzugeben. Die Angabe eines negativen Wertes bewirkt, daß vom Dateiende aus gezählt wird.

## 2.5 Druckeranweisungen

### HARDCOPY [H] Bildschirminhalt drucken

HARDCOPY

Der aktuelle Bildschirminhalt wird auf dem Drucker ausgegeben. Wird dieser Befehl ausgeführt, kann er nur durch gleichzeitiges Drücken der Tasten ALTERNATE/HELP unterbrochen werden. Nach ca. 30 Sekunden ist der Interpreter wieder arbeitsbereit. Ist der Drucker nicht angeschlossen, nicht angeschaltet oder nicht "On Line", wird die Ausführung des Befehls automatisch nach ca. 30 Sekunden abgebrochen.

### LLIST [LL] Programmlisting drucken

LLIST

Das im Arbeitsspeicher befindliche Programmlisting wird auf dem Drucker ausgegeben. Wird dieser Befehl ausgeführt, kann er nur durch Ausschalten, bzw. durch "Off Line"-Schalten des Druckers unterbrochen werden. Nach ca. 30 Sekunden ist der Interpreter wieder arbeitsbereit. Ist der Drucker nicht angeschloßen, nicht angeschaltet oder nicht "On Line", wird die Ausführung des Befehls automatisch nach ca. 30 Sekunden abgebrochen.

## LPOS Druckkopfposition ermitteln

**LPOS(Dummy)**

Bei Druckerausgaben wird eine bestimmte, vom jeweiligen Druckertyp abhängige Anzahl von Bytes in dessen Puffer gesammelt, bevor diese auf dem Drucker ausgegeben werden. Dieser Befehl ermöglicht nun, die Position zu ermitteln, auf welcher der virtuelle Druckkopf innerhalb dieses Puffers steht. Sie ist nicht unbedingt identisch mit der Position des realen Druckkopfes, der die Zeichen auf das Papier druckt.

## LPRINT [LPR] Daten an Drucker ausgeben

**LPRINT**
**LPRINT "Text"**
**LPRINT "Text"[;,']"Text"[;,']V_name[;,']V_name$[;,']...**

*LPRINT* gibt Daten und/oder Texte und/oder Leerzeilen (-zeichen) auf dem Drucker aus. Es gelten diesselben Ausführungen wie zu *PRINT*. Es ist jedoch nicht möglich, den Druckkopf mit Hilfe des Zusatzes *AT* zu positionieren. Mit den entsprechenden ESCAPE-Sequenzen (*CHR$(27)* siehe Druckerhandbuch) lassen sich auch hier verschiedene Steuerbefehle einsetzen.

## 2.6 Peripherieanweisungen

### CLOSE [CL] Datenkanal schließen

**CLOSE**
(schließt alle offenen Dateien)

**CLOSE #Kanalnummer**
(schließt die Datei mit dem angegebenen Identifikator)

*CLOSE* schließt entweder eine bestimmte oder alle vorher mit *OPEN* eröffneten Dateien. *Kanalnummer* enthält den Identifikator (0-99) der Datei, die geschlossen werden soll. Wird der Befehl ohne weitere Zusätze verwendet, werden alle offenen Dateien geschlossen. Im Falle, daß innerhalb des Programmlaufes keine Kanäle geschlossen und deren Nummer einer neuen Datei zugeordnet werden müssen, kann die Ausführung dieses Befehls entfallen, da bei Programmende durch *END*, *EDIT*, *SYSTEM* oder *QUIT* alle offenen Dateien automatisch wieder geschlossen werden.

### FIELD [FIE] Datensatz in Felder unterteilen

**FIELD #Kanalnummer,Anzahl AS V_name1$,Anzahl AS V_name2$...**

*FIELD* weist einer beliebigen Anzahl von Datenfeldern einzelne Bytelängen zu und füllt die für die Aufnahme der Felder vorgesehenen Stringvariablen mit dementsprechend vielen Leerzeichen. *Kanalnummer* enthält den Identifikator (0-99) der "R"-Datei (siehe *OPEN*), die in Datenfelder aufgeteilt werden soll. Es wird jeweils die in *Anzahl* enthaltenen Menge an Bytes einer nach *AS* folgenden Stringvariablen zugeordnet. Die mit *OPEN,"R", #Kanalnummer,"Dateiname",Satzlänge* eingerichtete Satzlänge wird so in verschiedene Datenfelder eingeteilt. Die Summe aller Feldgrößen sollte hier der angegebenen Satzlänge entspre-

chen. Bei mehreren Feldern sind die einzelnen Definitionen durch Kommata voneinander zu trennen. Es ist für jeden geöffneten Datenkanal im "R"-Modus nur eine FIELD-Anweisung zugelassen. Wird versucht mehrere FIELD-Anweisungen an denselben Datenkanal zu senden, erscheint eine Fehlermeldung. Die FIELD-Anweisung ist in ihrer Länge durch die maximale Eingabezeilenlänge von 256 Zeichen begrenzt.

## GET # — Datensatz lesen

**GET #Kanalnummer**
(liest den nächsten Datensatz)

**GET #Kanalnummer,Satznummer**
(liest den angegebenen Datensatz)

Liest einen Datensatz aus einer offenen "R"-Datei. *Kanalnummer* enthält den Identifikator (0-99) der "R"-Datei (siehe *OPEN*), aus welcher der angegebene Datensatz gelesen werden soll. Bei der Zuordnung von Datensätzen zu einer "R"-Datei wird jedem Satz eine Nummer zugeteilt. Unter Angabe dieser *Satznummer* kann derselbe Datensatz mit *GET#* wieder in die entsprechenden, mit *FIELD* spezifizierten Stringvariablen zurückgelesen werden. Wird die Angabe der *Satznummer* unterlassen, wird jeweils der nächste Datensatz gelesen.

## INP — Daten byteweise von Peripherie lesen

**INP (Port)**
**INP (#Kanalnummer)**

*INP* wartet auf die Übergabe eines einzelnes Bytes, das entweder von einer Diskettendatei oder von einem der Ports gesendet wird. In einer nachgestellten Klammer wird diesem Befehl entweder die Nummer des Datenkanals (0-99) oder eine Identifikationsnummer (*Port*) für die einzelnen Ports übergeben. Vorsicht: Dieser Befehl läßt sich auch durch die Abbruch-Funktion CONTROL / SHIFT /

ALTERNATE nicht widerrufen. Wurde er aufgerufen, ohne daß auf dem angegebenen Port ein Byte anliegt, muß das System neu gestartet werden.

Die Ports haben folgende Identifikatoren:

```
0 = Druckerport          (LST:)
1 = Auxiliary-Port       (AUX:)
2 = Consolport/Tastatur  (CON:)
3 = MIDI-Port            (MID:)
4 = Keyboard-Prozessor   (IKB:)
5 = Videoport/Monitor    (VID:)
```

**X=Inp(2)** wartet auf einen Tastendruck und speichert dessen ASCII-Wert in X.

**Print Inp(#1)**
gibt den ASCII-Wert des Zeichens einer offenen Datei mit der Kanalnummer #1 aus, auf dem sich der aktuelle Filepointer befindet.

## INPUT # [INP] Daten von Datenkanal lesen

**INPUT #Kanalnummer,V_name**
**INPUT #Kanalnummer,V_name1$,V_name2...**

*INPUT#* ermöglicht das Einlesen von Daten aus der angegebenen Datei. Diese Daten werden Variablen zugeordnet. *Kanalnummer* enthält den Identifikator(0-99) der Datei, aus welcher die Daten gelesen werden sollen. *V_name* steht hier für den Namen einer beliebigen Variablen, der die jeweilige Eingabe zugeordnet wird. Durch Kommata getrennt kann eine Liste von Variablen, auch unterschiedlichen Typs, angegeben werden. Die Variablen sind vom Typ her den zu lesenden Daten entsprechend anzugeben. Passen die Daten nicht zu der dafür vorgesehenen Variablen, erscheint eine Fehlermeldung. Bei Stringeingaben ist die Länge auf 255 Zeichen begrenzt. Zum formatgerechten Schreiben von Daten siehe *WRITE#*.

## LINE INPUT # [LI INPUT] Zeichenkette einlesen

**LINE INPUT #Kanalnummer,V_name$**

*LINE INPUT#* ermöglicht das Einlesen einer Zeichenkette aus der angegebenen Datei. Während beim INPUT#-Befehl die auftretenden Kommata als Trennzeichen zwischen den einzelnen Daten fungieren, werden hier Kommata als zum einzulesenden Text gehörig angenommen. *Kanalnummer* enthält den Identifikator (0-99) der Datei, aus welcher der String gelesen werden soll. *V_name$* steht hier für eine Stringvariable, welcher der gelesene String zugeordnet wird.

## OPEN # [O] Datenkanal öffnen

**Open "Modus",#Kanalnummer,"Dateiname"**

Es können Datenkanäle (Diskdateien, virtuelle Dateien) zum Einrichten, Schreiben, Lesen, Ergänzen oder Erneuern geöffnet werden.

**Modus:**

**O** (Output) - öffnet die Datei zum Lesen von Daten, oder, falls die angegebene Datei noch nicht vorhanden ist, installiert diese mit der angegebenen Kanalnummer.

**I** (Input) - öffnet eine vorhandene Datei zum Schreiben von Daten.

**A** (Append) - öffnet eine vorhandene Datei und setzt den Filepointer hinter das letzte Byte der Datei. Alle an diese Datei auszugebenden Daten werden an das Dateiende angehängt.

**U** (Update) - in diesem Modus kann gleichzeitig eine bestehende Datei geschrieben und gelesen werden.

**R** (Random) - es kann eine Random-Access-Datei eingerichtet werden. Mit Random-Access-Dateien ist es möglich, Datensätze für wahlfreien Zugriff innerhalb einer Datei anzulegen. Diese Datensätze werden mit den den Befehlen *GET#* und *PUT#* gelesen und geschrieben. Die Einrichtung der Datei erfolgt durch *FIELD#*. Der OPEN-Befehl erweitert sich gegebenenfalls hierfür um einen Parameter:

```
Open "R",#Kanalnummer,"Dateiname",Satzlänge
```

**Kanalnummer**

Gibt den Identifikator der zu bearbeitenden Datei an (max. 0-99). Soll eine bestimmte Datei durch entsprechende Befehle (*PRINT#*, *INPUT#* usw.) angesprochen werden, ist jeweils dieser Identifikator zu verwenden.

**Dateiname** Hierarchische Pfadstruktur (siehe Kapitelanfang) kann verwendet werden. Es können außerdem die folgenden virtuellen Dateien (Ports) zusätzlich angesprochen werden:

```
CON: = Consolport
LST: = Druckerport
PRN: = Druckerport
AUX: = Auxilliary-Port
MID: = MIDI-Ports
VID: = Monitor (Video)
IKB: = Tastaturprozessor
```

Der Ausdruck vor dem Gleichheitszeichen wird in diesem Fall als Dateiname an *OPEN* übergeben. *Modus* kann bei Verwendung dieser Möglichkeit entfallen.

## OUT [OU] Daten byteweise an Peripherie ausgeben

**OUT Port, Bytewert**
**OUT #Kanalnummer,Bytewert**

Es können einzelne Bytewerte an einen bestimmten Datenkanal oder einen Peripherie-Port gesendet werden. *Port* enthält den Identifikator des anzusprechenden Peripherie-Ports (siehe unter *INP*). Vorsicht: Werden Befehle an den "IKB:" (Intelligent Keyboard) gesendet, ist bei falscher Opcode-Angabe mit Absturz zu rechnen. *Kanalnummer* enthält den Identifikator (0-99) der Datei, an deren Filepointerposition der angegebene *Bytewert* geschrieben werden soll.

**Out 5,10** gibt das Zeichen LF (ASCII-Wert 10 - Steuerzeichen LINEFEED) an der aktuellen Cursorposition auf dem Monitor aus (es lassen sich also auch die Steuerzeichen *CHR$(0)* bis *CHR$(31)* auf dem Bildschirm beliebig plazieren).

**Out #1,13** schreibt ein CARRIAGE-RETURN an die aktuelle Filepointerposition in der Datei mit der angegebenen Kanalnummer.

## PRINT # [? oder P] Daten in Kanal schreiben

**PRINT #Kanalnummer**
**PRINT #Kanalnummer,"Text"**
**PRINT #Kanalnummer, "Text" [;,'] "Text" [;,'] V_name [;,'] V_name$[;,']...**

Der Befehl schreibt Daten und/oder Texte und/oder Leerzeilen (-zeichen) in eine geöffnete Datei. Er erfüllt die gleiche Funktion wie *PRINT*, nur das hiermit die Daten in einen geöffneten Datenkanal geschrieben werden können. *Kanalnummer* steht für den Identifikator (0-99) der Datei, in welche die Daten und Texte geschrieben werden sollen. Sonst siehe unter *PRINT*.

## PRINT #, USING — Daten formatiert schreiben

**PRINT #Kanalnummer,USING "format", Liste**

Es können Strings und Zahlen in einem vorgegebenen Format an eine geöffnete Datei ausgegeben werden. Dieser Befehl erfüllt die gleiche Funktion wie *PRINT USING*, nur das hiermit die Daten in einen geöffneten Datenkanal geschrieben werden können. *Kanalnummer* steht für den Identifikator (0-99) der Datei, in welche die formatierten Datenausgaben geschrieben werden sollen. Zur Formatangabe siehe unter *PRINT USING*.

## PUT # [PU] — Datensatz schreiben

**PUT #Kanalnummer**
(schreibt den nächsten Datensatz)

**PUT #Kanalnummer,Satznummer**
(schreibt den angegebenen Datensatz)

*PUT#* schreibt einen Datensatz in eine geöffnete "R"-Datei. *Kanalnummer* enthält den Identifikator (0-99) der "R"-Datei (siehe *OPEN*), in welche der angegebene Datensatz geschrieben werden soll. Bei der Zuordnung von Datensätzen zu einer "R"-Datei wird jedem Satz eine Nummer zugeteilt. Unter Angabe dieser *Satznummer* kann ein definierter Datensatz mit *PUT#* aus den entsprechenden, mit *FIELD* spezifizierten Stringvariablen in die Datei geschrieben werden. Um die zuzuordnenden Texte formatgerecht in die FIELD-Variablen einzufügen, eignen sich die Befehle *LSET* und *RSET*. Die Satznummer ist so zu wählen, daß Sie nicht größer ist, als die in der entsprechenden Datei enthaltenen Datensätze. Maximal ist eine Datensatzanzahl pro Datei von 65535 möglich. Wird die Angabe der *Satznummer* unterlassen, wird jeweils der nächste Datensatz gelesen.

## WRITE # [WR] Daten sequentiell schreiben

**WRITE #Kanalnummer**
**WRITE #Kanalnummer",Text"**
**WRITE #Kanalnummer,"Text",V_name,V_name$...**

Der eigentliche Sinn dieses Befehl ist das formatgerechte, auf den INPUT#-Befehl zugeschnittene Speichern von Werten und Texten. Sollen mit dem INPUT#-Befehl mehrere Werte oder Strings gleichzeitig eingelesen werden, so müssen die Einzeldaten durch Kommata voneinander getrennt sein. Da der WRITE#-Befehl auch Kommata an den entsprechenden Platz in der Datei schreibt, können die Daten beim Einlesen mit *INPUT#* unterschieden und den dabei angegebenen Variablen zugeordnet werden. *Kanalnummer* enthält den Identifikator (0-99) der Datei, in welche die Daten geschrieben werden sollen. *"Text"*, *V_name* und *V_name$* stehen für die verschiedenen Möglichkeiten, eine durch Kommata getrennte Ausdrucksliste zu bilden.

## 2.7 Soundchip-Anweisungen

### SOUND [SO] Klangausgabe

**SOUND Kanal,Volume,Note,Oktave,Dauer**
**SOUND Kanal,Volume,#Periode,Dauer**

Der Soundchip bietet die Möglichkeit, 3 Klangausgabe-Kanäle anzusprechen. Dies bedeutet, daß 3 verschiedene Töne oder Geräusche gleichzeitig generiert werden können. Dazu müssen die Klangkanäle mit dem WAVE-Befehl "eingerichtet" werden, anderenfalls ist eine mehrstimmige Klangausgabe nicht möglich.

Der SOUND-Befehl benötigt 4 bzw. 5 Parameter:

**Kanal** ist die Nummer des angewählten Ausgabekanals (1,2,3).

**Volume** regelt die Lautstärke in der Abstufung von 0 (ausgeschaltet) bis 15 (laut).

Zur Tonhöhen-Bestimmung gibt es 2 verschiedene Möglichkeiten:

a) Chromatisch:

*Note* bestimmt die "musikalische Tonhöhe" der Klangausgabe:

```
1   2   3   4   5   6   7   8   9   10  11  12
-------------------------------------------
C   Cis D   Dis E   F   Fis G   Gis A   B   H
    (Des) (Es)      (Ges)   (As)    (Ais)
```

*Oktave* bestimmt die Oktave, in welcher der gewünschte Ton ausgegeben werden soll. Mit 1 bestimmt man die tiefste, mit 8 die höchste Oktave.

b) Numerisch:

*Periode* bestimmt die "physikalische Tonhöhe" der Klangausgabe. Es wird ein Wert mit vorangestelltem # eingesetzt, der aus dem Teilungsergebnis von maximaler Ausgabefrequenz (125000 Hertz) durch die gewünschte Frequenz entsteht (als Ganzzahl): *#Periode = TRUNC(125000 / Frequenz +0.5)*

Es kann ein Ton somit auf 2 verschiedene Arten erzeugt werden. z.B. Kammerton A , 2 Sekunden lang

```
SOUND 1,15,10,4,100      (10.er Ton, 4. Oktave)
SOUND 1,15,#284,100      (Trunc(125000 Hz / 440 Hz +0.5)
```

*Dauer* bestimmt, wie lange abgewartet wird, bevor der nächste Befehl ausgeführt wird. Der Ton wird für die angegebene Zeitdauer (Schritte in 1/50 Sekunden) gehalten.

## WAVE [WA] Soundregister einstellen

**WAVE Kanal,Hüllkurve,Kurvenform,Periode,Dauer**

*WAVE* ermöglicht den Zugriff auf die maßgebenden Register des Soundchips und somit die Klangbeeinflussung und die Mehrstimmigkeit. Damit *WAVE* eine Wirkung erzielt, muß der Soundchip erstmal mit dem SOUND-Befehl angeschaltet werden. *Kanal* bestimmt, welche Ausgabekanäle angesprochen werden. Im Gegensatz zu *SOUND* wird hier ein Wert eingetragen, der sich bitweise zusammenaddiert aus:

```
1  = Bit 0   (Kanal 1)
2  = Bit 1   (Kanal 2)
4  = Bit 2   (Kanal 3)
8  = Bit 3   (zusätzlich Rauschen auf Kanal 1)
16 = Bit 4   (zusätzlich Rauschen auf Kanal 2)
32 = Bit 5   (zusätzlich Rauschen auf Kanal 3)
```

*Hüllkurve* bestimmt, welcher Kanal (incl. Rauschen) von der Hüllkurve moduliert (verändert) wird. Dieser Wert setzt sich auch bitweise zusammen und wird, wie unter *Kanal* beschrieben, angegeben. Mit dem Parameter *Kurvenform* wird die Art der Hüllkurve definiert:

| | |
|---|---|
| Linear fallend | = 0,1,2,3,9 |
| Linear fallend, sprunghaft | = 11 |
| Linear steigend, abbrechend | = 4,5,6,7,15 |
| Linear steigend, haltend | = 13 |
| Sägezahn fallend | = 8 |
| Sägezahn steigend | = 12 |
| Dreieck, Beginn fallend | = 10 |
| Dreieck, Beginn steigend | = 14 |

*Periode* ist ein numerischer Wert, der die Länge der Hüllkurve festlegt. Wird er kleiner, so staucht sich die Hüllkurve (bei graphischer Darstellung) und die Modulationsgeschwindigkeit erhöht sich.

*Dauer* bestimmt, wie lange gewartet wird, bis der nächste Befehl ausgeführt wird. Wird ein Klang angeregt, so schaltet sich der Soundchip erst ab, wenn er durch den Befehl WAVE 0,0 dazu veranlaßt oder eine beliebige Taste gedrückt wird.

# 3. Programmstrukturen

## 3.1 Schleifenkonstruktionen

### DO...LOOP [DO...L] Endlosschleife

**DO**
...auszuführende Programmteile...
**LOOP**

Eine DO...LOOP-Schleife kann nur abgebrochen werden, wenn sie auf eine Abbruchanweisung (*END*, *EDIT*, *STOP*) trifft, eine EXIT IF-Anweisung findet und die Abbruchbedingung erfüllt wird oder eine GOTO-Anweisung innerhalb der Schleife zu einem Label außerhalb der Schleife verzweigt. Es ist ebenfalls möglich, die Schleife durch eine GOSUB-Anweisung zu verlassen und aus der angesprochenen Prozedur wieder in die Schleife zurückzukehren. Außerdem kann zum Abbruch jederzeit die übliche Abbruchfunktion CONTROL / SHIFT / ALTERNATE verwendet werden. DO...LOOP-Schleifen können beliebig oft ineinander verschachtelt werden.

### FOR...NEXT [F...N] Zählschleife

**FOR Zähler = Anfang TO/DOWNTO Ende STEP Schritt**
...auszuführende Befehle...
**NEXT Zähler**

Erstellt eine Schleife, die so oft ausgeführt wird, wie in der Schleifenbedingung vorgegeben. Die Kopfzeile der Schleife enthält den Anfangswert *Anfang* und den Endwert *Ende*. Die Zählvariable *Zähler* wird, beginnend mit *Anfang* solange erhöht oder vermindert, bis sie den Wert *Ende* erreicht hat. Danach wird das Programm mit der auf die NEXT-Anweisung folgenden Programmzeile fortgesetzt. Wird nur *FOR TO/NEXT* ohne Zusätze (*DOWNTO/STEP*)

verwendet, beträgt die Schrittweite +1. Bei Verwendung von *DOWNTO* ist der Anfangswert größer anzugeben als der Endwert, da in diesem Fall die Schrittweite -1 beträgt. Der optionale Zusatz *STEP* bewirkt, daß der nach *STEP* angegebene Wert oder Ausdruck als Schrittweite angenommen wird. Es kann hier auch ein negativer Wert übergeben werden. Bei jeder Erhöhung oder Verminderung der Zählvariablen werden die Programmzeilen zwischen der Kopfzeile und dem durch *NEXT* gekennzeichneten Schleifenende einmal ausgeführt. FOR..NEXT-Schleifen können beliebig oft ineinander verschachtelt werden, solange den einzelnen Zählvariablen verschiedene Namen zugeordnet sind. Wenn es irgend möglich ist, sollte für die Zählvariable eine Integervariable verwendet werden, da sich die Durchlaufzeit dadurch bei kleinen und nicht so arbeitsaufwendigen Schleifen erheblich verkürzt. Außerdem benötigen Integervariablen grundsätzlich nur 2/3 des Speicherplatzes einer Fließkommavariablen.

## REPEAT...UNTIL [REP...U] Bedingte Schleife

**REPEAT**
...auszuführende Programmteile...
**UNTIL Bedingung**

Bildet eine Schleife mit implizierter Abbruchbedingung am Schleifenende. Die Bedingung zum Verlassen der Schleife wird am Schleifenende geprüft. Das heißt, daß die Schleife mindestens einmal bis zu der in *UNTIL* vereinbarten Bedingung durchlaufen wird. Ist die Abbruchbedingung erfüllt, wird das Programm mit der nächsten auf *UNTIL* folgenden Zeile fortgesetzt. REPEAT...UNTIL-Schleifen können beliebig oft ineinander verschachtelt werden.

## WHILE...WEND [W ... WE] Bedingte Schleife

**WHILE Bedingung**
...auszuführende Programmteile...
**WEND**

Die Bedingung zum Verlassen der Schleife wird am Schleifenanfang geprüft. Das heißt, daß die Schleife nicht durchlaufen wird, wenn die bei *WHILE* vereinbarte Bedingung erfüllt ist. Ist die Abbruchbedingung erfüllt, wird das Programm mit der nächsten auf *WEND* folgenden Zeile fortgesetzt. WHILE...WEND-Schleifen können beliebig oft ineinander verschachtelt werden.

## 3.2 Kontrollbefehle

### EXIT IF [E IF] Bedingter Schleifenabbruch

**EXIT IF Bedingung**

Bricht den Schleifendurchlauf unabhängig von der zu der jeweiligen Schleife gehörigen Abbruchbedingung ab. Dieser Befehl ist eine Sonderfunktion, um in FOR/NEXT-, DO/LOOP-, REPEAT/UNTIL- und WHILE/WEND-Schleifen an bestimmten Stellen in dem zwischen Einstiegs- und Wendepunkt eingeschlossenen Programmblock zusätzliche Abbruchbedingungen zu formulieren. Es kann unabhängig vom Zustand der Schleife diese jederzeit verlassen werden, wenn die durch *EXIT IF* gestellte Bedingung mit "wahr" beantwortet wird. Ist dies der Fall, wird das Programm mit der nach dem Schleifenwendepunkt stehenden Programmzeile fortgesetzt.

### GOSUB [GO oder @] Zu Prozedur verzweigen

**GOSUB Procedurename**
**GOSUB Procedurename(Parameterliste)**

Es kann zu einer definierten Prozedur verzweigt und dieser eine Parameterliste übergeben werden. *Procedurename* steht für den Namen der Prozedur, zu welcher verzweigt werden soll. Wird die optionale Angabe einer *Parameterliste* verwendet, kann diese beliebig lang sein. Es ist darauf zu achten, daß die Anzahl der Parameter mit der Anzahl der in der entsprechenden Prozedur benannten Aufnahmevariablen und die Datentypen der Liste mit den Variablentypen übereinstimmen. Die Datentypen können innerhalb der Liste beliebig gemischt werden, solange die gleiche Typenfolge in der Liste der Aufnahmevariablen eingehalten wird.

## GOTO [GOT] Verzweigung zu einem Label

**GOTO Label**

Durch *GOTO* kann zu einer beliebigen Programmstelle verzweigt werden. Es gibt hier keine implizite Rückkehrmöglichkeit. Das Programm wird also von der angesprungenen Zielzeile an fortgesetzt. *Label* ist ein beliebig bennenbarer Name, der zur Markierung eines Sprungziels für GOTO- oder RESTORE-Anweisungen (siehe *RESTORE*) dient. Wie bei Prozedurnamen ist es auch hier möglich, eine Ziffer als erstes Zeichen für einen Labelnamen zu verwenden. Er kann an jeder beliebigen Stelle im Programm plaziert werden. Zur Unterscheidung von Variablennamen ist er mit einem nachfolgenden Doppelpunkt (*Label:*) zu kennzeichnen. GOTO-Sprünge in oder aus FOR/NEXT-Schleifen oder Prozeduren sind nicht möglich.

## IF (ELSE) ENDIF [I..EL..EN] Bedingungsabfrage

**IF Bedingung**
...auszuführende Programmteile, wenn die Bedingung erfüllt ist...
**ELSE**
...auszuführende Programmteile, wenn die Bedingung nicht erfüllt ist...
**ENDIF**

Es ist möglich, umfangreiche Programmteile von der Erfüllung einer einzigen Bedingung abhängig zu machen. Die Folge von Befehlen, die abhängig von *IF* ausgeführt werden sollen, werden ausschließlich von der zugehörigen ENDIF-Bestimmung eingegrenzt. Wird der optionale Zusatz *ELSE* verwendet, sind zwischen diesem und dem zugehörigen *ENDIF* die Programmteile anzugeben, die ausgeführt werden sollen, wenn sich die IF-Abfrage als unwahr erweist. IF-Abfragen können beliebig ineinander verschachtelt werden.

## ON BREAK — Unterbrechungen

**ON BREAK GOSUB Procedurename**
**ON BREAK CONT**

Bewirkt die Verzweigung zu einer definierten Prozedur, falls während des Programmlaufs die Abbruchtasten-Kombination CONTRL / SHIFT / ALTERNATE gedrückt wird. Der Befehl kann an jeder beliebigen Stelle im Programm (auch mehrfach) verwendet werden. Es ist also möglich, auf Abbruchversuche des Anwenders je nach Situation unterschiedlich zu reagieren, oder, durch den Zusatz *CONT*, das Programm ohne Unterbrechung fortzusetzen.

## ON ERROR GOSUB — Fehlerverzweigung

**ON ERROR GOSUB Procedurename**
(Verzweigung bei Fehler)

**ON ERROR**
(Ausgabe der Interpreter-Meldungen)

Verzweigt zur angegebenen Prozedur, sobald ein Fehler auftritt, bzw. schaltet die Ausgabe interpretereigener Fehlermeldungen wieder ein. *Procedurename* steht für den Namen einer beliebig zu benennenden Prozedur, zu welcher das Programm verzweigen soll, sobald ein Fehler auftritt. Das Programm verzweigt ohne Unterbrechung dorthin. *ON ERROR GOSUB* ist immer vor der evtl. Fehlerstelle einzusetzen. Wurde zu einer programminternen Fehlerroutine verzweigt, schaltet der Interpreter die Fehlerbehandlung nach Abarbeiten der Prozedur wieder in den Normalmodus zurück. Ist kein Fehler aufgetreten und soll der Normalmodus wieder eingeschaltet werden, wird der Befehl *ON ERROR* verwendet. Der Interpreter meldet anschließend evtl. auftretende Fehler wieder mit der Errorbox, verzweigt nicht mehr zu der angegebenen Routine und unterbricht das Programm.

## ON...GOSUB Verzweigung zu Prozeduren

**ON Wert GOSUB Procedure1,Procedure2,Procedure3,...**

*ON GOSUB* trifft die Entscheidung darüber, zu welcher Prozedur verzweigt werden soll, in Einerschritten ab 1 aufwärts. Eine Wertestaffelung, die mit einem Wert >1 beginnt, ist nicht möglich. Die Verzweigung erfolgt nach folgendem Schema:

```
Wert  >= 1 < 2  -->  Procedure1
Wert  >= 2 < 3  -->  Procedure2
Wert  >= 3 < 4  -->  Procedure3 usw.
```

Tritt ein *Wert* auf, der größer ist, als die Anzahl der angegebenen Prozeduren oder ist *Wert* gleich 0, wird das Programm in der auf *ON..GOSUB* folgenden Programmzeile fortgesetzt.

## RESUME [RESU] Programm fortsetzen

**RESUME**
Nach Error-Routine Programmfortsetzung mit Wiederholung der Zeile, in welcher der Fehler aufgetreten ist.

**RESUME NEXT**
Nach Error-Routine Programmfortsetzung mit der, auf die fehlerhafte Zeile folgenden Programmzeile

**RESUME Label**
Nach Error-Routine Programmfortsetzung mit der Programmzeile, welche auf die mit dem angegebenen Labelnamen versehenen Zeile folgt.

Bestimmmt, ab welcher Programmzeile das Programm nach Auftreten eines selbst verwalteten Fehlers fortgesetzt werden soll. Wird *RESUME* ohne weitere Zusätze verwendet, springt das Programm von der programmeigenen Error-Routine zu der Zeile zurück, in welcher der Fehler aufgetreten ist und wiederholt diese. *RESUME NEXT* bewirkt, daß die auf die fehlerhafte Zeile

folgende Programmzeile als Sprungziel bestimmt und das Programm ab dieser Zeile fortgesetzt wird. Befindet sich das angegebene Label außerhalb der Error-Routine, wird das Register, in dem die GOSUB-Sprünge festgehalten werden, gelöscht und alle globalen Variablen wieder restauriert. Bei evtl. auftretenden Bomben-Errors (Err = 102-109) können *RESUME* und *RESUME NEXT* nicht eingesetzt werden. In diesem Fall muß *RESUME LABEL* verwendet werden.

## TRON — Debugging einschalten

**TRON**
**TRON #Kanalnummer**

Listet die aktuell ausgeführten Programmzeilen während des Programmlaufs auf dem Bildschirm, Drucker oder in einer Diskettendatei auf. Wird der Befehl ohne den Zusatz *Kanalnummer* verwendet, werden die aktuellen Programmzeilen auf dem Monitor ausgegeben. *Kanalnummer* enthält den Identifikator der Datei (0-99), in welche das Listing geschrieben werden soll. Die Datei ist dazu vorher mit *OPEN* zu eröffnen (siehe *OPEN*).

## TROFF — Debugging ausschalten

**TROFF**

*TROFF* beendet die mit *TRON* begonnene Programmüberwachung. kann, wie *TRON*, an jeder beliebigen Stelle des Programms eingesetzt werden. Im Anschluß an diesen Befehl ist das Programm wieder im Normalmodus. Wurden die Programmzeilen mit *TRON #Kanalnummer* in eine Datei geschrieben, kann die Angabe der *Kanalnummer* bei *TROFF* unterbleiben.

## 3.3 Strukturbefehle

### CLEAR [CLE] Felder und Variablen löschen

**CLEAR**

Alle numerischen Variablen erhalten den Wert 0, alle Zeichenketten-Variablen werden zu Leerstrings. Felder werden gelöscht und ihre Dimensionierung aufgehoben. Der CLEAR-Befehl darf nicht in Prozeduren oder in FOR...NEXT-Schleifen vorkommen, da diese ohne eine Variablenbelegung (z.B Zählvariable/Stapelzeiger) nicht funktionieren. Am Programmanfang ist der CLEAR-Befehl nicht nötig, da durch *RUN* alle Variablen gelöscht werden.

### CLS Bildschirm löschen

**CLS**
**CLS #Kanalnummer**

Wird der Befehl ohne weitere Zusätze eingesetzt, wird der Ausgabebildschirm bzw. das jeweils geöffnete GEM-Fenster gelöscht und der Cursor in die linke obere Ecke des Bildschirms (*HOME*), bzw. des Fensters gesetzt. Mit *CLS#* können auch Diskettendateien oder virtuelle Dateien unter Angabe der entsprechenden *Kanalnummer* angesprochen werden (siehe unter *OPEN*). Im Falle, daß *CLS#* für eine Diskettendatei verwendet wird, wird beim Lesen dieser Datei der Bildschirm gelöscht, sobald der Filepointer auf diesen Befehl innerhalb der Datei trifft.

## CONT [CON] Programm nach STOP fortsetzen

**CONT**

Wurde der Programmlauf mit *STOP* unterbrochen, kann durch die Eingabe von *CONT* im Direktmodus das Programm in der Zeile nach dem STOP-Befehl fortgesetzt werden. *CONT* kann nicht eingesetzt werden, wenn nach dem Programmabbruch entweder der CLEAR-Befehl verwendet, das Programmlisting verändert oder neue Variablen eingeführt wurden.

## DATA [D] Datenspeicher

**DATA Eintrag1,Eintrag2,"Texteintrag1",Texteintrag2...**

Es wird eine Programmzeile als Werte-, bzw. Textspeicher deklariert. Der Anweisung wird eine Liste durch Kommata getrennter Werte oder Texte übergeben. Sie dient dazu, einem auftretenden READ-Befehl die entsprechende Anzahl von Daten zur Verfügung zu stellen. Wenn in Text-Datas keine Kommata berücksichtigt werden müssen, müssen Texte nicht in Anführungszeichen gesetzt werden. Der Interpreter liest in diesem Fall alle Zeichen (auch Leerzeichen), die zwischen den einschließenden Kommata aufgeführt sind. Für den Interpreter ist es unerheblich, wenn bei alphanumerischen READ-Anweisungen numerische Angaben gefunden werden. Anders verhält es sich bei numerischen READ-Anweisungen. Diese sind darauf angewiesen, auch numerische Datas vorzufinden. Die Datas können dann allerdings auch in der binären, hexadezimalen oder oktalen Schreibweise angegeben sein. Variablen können in den Datas nicht übergeben werden. Kommen im Programm keine READ-Anweisungen vor, kann in Data-Zeilen beliebiger Text stehen. Dieser Text bleibt dann wie bei *REM* vom Programm unberücksichtigt.

**DEFFN Funktionsname=Funktionsausdruck**
**DEFFN Funktionsname(Variablenliste)=Funktionsausdruck**

Es können mathematische oder alphanumerische Funktionen kompakt definiert werden. *Funktionsname* steht für einen beliebig zu wählenden Namen, mit welchem die Funktion durch *FN* angesprochen werden kann. Dieser Name hat die Funktion einer Variablen, welcher der in *Funktionsausdruck* ermittelte Wert zugewiesen wird. Zur Namensbildung können auch bereits bestehende Variablennamen oder BASIC-Befehlsnamen verwendet werden. Das erste Zeichen kann, anders als bei normalen Variablen, auch eine Ziffer sein. Mit dem optionalen Zusatz *Variablenliste* können mehrere Variablen angegeben werden, denen ein evtl. mit *FN* übergebener Wert zugeordnet wird. Diese Variablenliste kann auch Variablen verschiedener Typen beinhalten, solange die mit *FN* zu übergebenden Daten in ihrer Reihenfolge zum gleichen Typ gehören. Innerhalb der Funktion sind nur Operationen erlaubt, die dem verwendeten Funktionstypen entsprechen. D.h., wenn die Funktion als Stringtyp deklariert wurde (z.B. *DEFFN Funktion$*), sind nur Stringoperationen möglich.

Beim Aufruf der Funktion werden die aktuellen Inhalte der darin verwendeten Variablen angenommen. Die Länge einer Funktion wird durch die maximale Eingabezeilenlänge (256 Zeichen) beschränkt. Reicht diese Länge nicht aus, können aus einer Funktion heraus andere Funktionen aufgerufen werden. Funktionen können an jeder beliebigen Stelle des Programms definiert sein. Da bei Programmstart alle *DEFFN* initialisiert werden, kann dies also auch am Programmende geschehen, selbst wenn der Funktionsaufruf *FN* vor der Funktion auftritt. Eine "Endlosschleife", worin sich zwei Funktionen gegenseitig aufrufen, kann auch durch die Abbruchfunktion CONTRL/SHIFT/ ALTERNATE nicht unterbrochen werden. Rekursive Aufrufe haben denselben Effekt.

## DEFLIST [DEFLIS] Listingformat festlegen

**DEFLIST X**

*DEFLIST* bestimmt das optische Erscheinungsbild des Programmlistings. Der GfA-BASIC-Editor kann ein Programmlisting in 2 Varianten darstellen:

x <> 0 alle Anfangsbuchstaben von BASIC-Befehlsnamen, sowie von Variablennamen werden groß geschrieben. (*PRINT A,B, Variable, LEFT$ ("Text")*)

x = 0 alle BASIC-Befehlsnamen werden groß geschrieben; alle Variablennamen werden klein geschrieben (*PRINT a,b, Variable, LEFT$ ("Text")*)

Der Übergabeparameter X bestimmt das Listing. Die erste Darstellungsart ist beim Interpreterstart voreingestellt. Der Befehl kann nur im Direktmodus verwendet werden.

## EDIT [ED] Programm beenden

**EDIT**

*EDIT* bewirkt einen Programmabbruch, löscht alle Variablen und schließt alle offenen Dateien. Dieser Befehl ist völlig identisch mit dem Befehl *END*, mit der Ausnahme, daß der GfA BASIC-Interpreter bei *EDIT* keine Programmende-Meldung ausgibt, sondern sofort zum Editor zurückkehrt. Der Abbruchbefehl schließt den Programmlauf vollständig ab. Er kann danach nicht durch *CONT* fortgesetzt werden. Es werden alle noch offenen #-Dateien geschlossen und alle Programmvariablen gelöscht. Im Direktmodus erlaubt *EDIT* (Neben <ESC>) die Rückkehr zum Editor.

## END — Programm beenden

**END**

Das Programm wird an der Stelle abgebrochen, an welcher es auf die END-Anweisung trifft. Es erscheint eine Programmende-Meldung, nach welcher zum Editor zurückgekehrt wird. Der Abbruchbefehl schließt den Programmlauf vollständig ab. Er kann danach nicht durch *CONT* fortgesetzt werden. Es werden alle noch offenen #-Dateien geschlossen und alle Programmvariablen gelöscht.

## FN [@] — Funktion aufrufen

**FN Funktionsname**
**FN Funktionsname(Parameterliste)**

*FN* ruft eine mit *DEFFN* definierte Funktion auf. Wurde bei *DEFFN* eine Variablenliste vorgegeben, müßen beim Funktionsaufruf in *Parameterliste* durch Kommata getrennt dementsprechend viele und dem jeweiligen Variablentyp entsprechende Werte, Strings, andere Variablen oder Ausdrücke übergeben werden. Entsprechen die übergebenen Parametertypen nicht der Liste oder werden zuviel, bzw. zuwenig Parameter übergeben, erscheint eine Fehlermeldung. Die Inhalte einer Funktion können entweder direkt ausgegeben, einer dem Funktionstyp entsprechenden Variablen übergeben oder in Bedingungsabfragen eingebunden werden.

## LET [LE] Daten zuweisen

**LET V_name=Ausdruck**
**LET V_name$=Textausdruck**

*LET* weist einer beliebigen Variablen Daten bzw. Variablenbelegungen zu. *Ausdruck* kann ein Wert oder eine beliebige numerische, alphanumerische, boolsche oder Stringvariable sein, die vom selben Typ wie die Zuweisungsvariable *V_name* oder *V_name$* sein muß. Prinzipiell kann eine Wertezuweisung auch allein mit dem Gleichheitszeichen (=) vorgenommen werden. Der Befehl *LET* ist also optional. Die Variable vor dem Gleichheitszeichen erhält den Wert des *Ausdrucks* hinter dem Gleichheitszeichen. Die Benutzung des Zusatzes *LET* birgt den Vorteil in sich, daß die Variable *V_name* mit einem beliebigen Namen versehen werden kann. Da es im GfA-BASIC nur wenige endgültig reservierte Worte gibt (z.B. *PI*, *ERR*, *FATAL*, *TIMER*, *TIME$*, *DATE$*), können auch BASIC-Befehlsnamen durch *LET* als Variablennamen benutzt werden.

## LOCAL [LOC] Lokale Variablen deklarieren

**LOCAL Lv_name**
(lokale Variable)

**LOCAL Lv,Lv%,Lv$...**
(lokale Variablenliste)

Es können innerhalb einer Prozedur Variablen als lokal deklariert werden. Lokale Variablen haben die Eigenschaft, daß sie ausschließlich in der Prozedur verwendet werden können, in der sie deklariert wurden. Wird außerhalb dieser Prozedur eine Variable mit gleichem Namen benannt, so unterscheidet der Interpreter diese von den prozedurinternen Variablen . Es können also außerhalb von Prozeduren Variablen mit denselben Namen benannt werden, wie schon deklarierte lokale Variablen, da der Interpreter lokale Variablen nur im direkten Zusammenhang mit der entsprechenden Prozedur erkennt. Es kann dem Befehl eine Liste von Variablen unterschiedlicher Typen übergeben werden.

## NEW — Programmspeicher löschen

NEW

Der Befehl *NEW* löscht an jeder beliebigen Stelle eines Programms den BASIC-Arbeitsspeicher, d.h. es wird das Programm mitsamt seinen Variablen und allen Initialisierungen gelöscht. Der Speicherplatz ist nun für eine neue Anwendung voll verfügbar.

## PAUSE [PA] — Wartefunktion

**PAUSE Dauer**

*PAUSE* bewirkt eine genau definierbare Pause. Der Parameter *Dauer* bestimmt, wie lange das Programm unterbrochen werden soll. Diese Angabe wird in 1/50 Sekunden eingegeben (1 Sekunde warten entspricht Pause 50).

## PROCEDURE [PRO] — Prozeduranfang

**PROCEDURE Name**
**PROCEDURE Name(Lv_name%,Lv_name,Lv_name$...)**

*PROCEDURE* definiert den Anfang einer Prozedur und stellt gegebenenfalls eine Liste lokaler Variablen zur Verfügung. *Name* kann ein beliebiger Name sein, der die Prozedur benennt. Als Bestandteile dieses Namens sind alle verfügbaren Zeichen zulässig. Im Gegensatz zu Variablennamen kann ein Prozedurname auch mit einer Ziffer beginnen. Wird der Prozedur eine Variablenliste zugeordnet, welche die mit dem entsprechenden GOSUB-Befehl übergebenen Daten aufnehmen sollen, ist diese in Klammern dem Prozedurnamen anzufügen. Innerhalb dieser Liste ist es unerheblich, ob alle Variablen vom gleichen Typ sind, solange die mit einem GOSUB-Befehl übergebenen Daten in ihrer Reihenfolge den Typen der Variablenliste entsprechen. Die in der Liste enthaltenen Variablen gelten innerhalb der dazugehörigen Prozedur als lokale Variablen

(siehe *LOCAL*). D.h., daß die in ihnen enthaltenen Werte oder Strings nur innerhalb dieser Prozedur eingesetzt oder erfragt werden können. Sollen innerhalb der Prozedur mit lokalen Variablen verschiedene arithmetische oder sprachliche Konstrukte gebildet werden, müssen diese dort vor Verlassen der Prozedur in globale Variablen übergeben werden, um ihre Inhalte außerhalb der Prozedur weiterverwenden zu können. GfA-BASIC bietet weiterhin die Möglichkeit zu rekursiven Aufrufen von Prozeduren. Es können somit also Prozeduren auch aus sich selbst aufgerufen werden. Jede Prozedur wird mit *RETURN* (siehe dort) abgeschlossen.

## QUIT [Q] Interpreter verlassen

**QUIT**

*QUIT* ist identisch mit *SYSTEM* und bewirkt, daß der Interpreter das Programm beendet und ohne eine Sicherheitsabfrage zum Desktop zurückkehrt.

## READ [REA] DATA-Werte auslesen

**READ V_name**

Dient dazu, die relevanten Data-Einträge zu lesen und sie gleichzeitig der angegebenen Variablen zuzuordnen. *V_name* steht für den Namen einer Variablen, welcher der jeweils gelesene Data-Eintrag zugeordnet werden soll. Wie bei *RESTORE* beschrieben, kann ein Zeiger auf ein bestimmtes Label gerichtet werden. *READ* liest die Data-Einträge, die auf das angegebene Label folgen und ordnet sie der jeweils angegebenen Variablen zu. Wird kein *RESTORE* verwendet, werden der Reihe nach vom Programmanfang aus soviele Datas eingelesen, wie READ-Anweisungen vorhanden sind. Sind weniger Datas vorhanden als READ-Anweisungen, wird eine entsprechende Fehlermeldung ausgegeben.

## REM [R oder '] Kommentar einfügen

**REM**
**REM Kommentar**

Es kann eine beliebige Kommentarzeile an jeder beliebigen Stelle des Programms eingesetzt werden.

## RESTORE [RES] DATA-Zeiger setzen

**RESTORE**
(richtet den Read-Data-Zeiger auf die erste aller Data-Zeilen.)

**RESTORE Label**
(richtet den Read-Data-Zeiger auf die Zeile, die mit *Label:* gekennzeichnet ist)

*RESTORE* restauriert den Data-Speicher bzw. richtet den Read-Data-Zeiger auf die dem angegebenen Label folgende Data-Zeile. *RESTORE* ohne Angabe eines Labelnamens bewirkt, daß der READ-DATA-Zeiger auf die erste, im Programm auftretende DATA-Zeile gerichtet wird. Alle folgenden READ-Anweisungen beziehen ihre Daten nacheinander ab dieser Zeile. Wird dem Befehl ein Labelname übergeben, zeigt der READ-DATA-Zeiger auf den Anfang der Data-Zeile, die auf den angegebenen Labelnamen folgt. Ab dort werden dann die gewünschten Datawerte eingelesen. Es müssen dann bis zur letzten Zeile des Programms noch so viele Datawerte vorhanden sein, wie mit *READ* gelesen werden. Ist dies nicht der Fall, wird eine Fehlermeldung ausgegeben.

## RETURN [RET] Prozedurschluß

RETURN

Bildet den Abschluß einer Prozedur und veranlaßt den Interpreter mit der nächsten Zeile nach dem Prozeduraufruf fortzufahren.

## RUN [RU] Programm starten

RUN

Der Befehl *RUN* initialisiert und startet das im Interpreter vorhandene BASIC-Programm von der ersten Zeile an. Dabei werden sämtliche Variablen und der Bildschirm gelöscht. *RUN* kann im Programm enthalten sein oder im Direktmodus eingegeben werden.

## STOP [ST] Programm unterbrechen

STOP

Mit *STOP* kann der Programmlauf an jeder Stelle unterbrochen werden. Es werden keine Variablen gelöscht, keine Dateien geschlossen (wie bei *END*) und das Programm kann durch Eingabe von *CONT* im Direktmodus fortgesetzt werden. (siehe unter *CONT*)

## SYSTEM [SYS] Interpreter verlassen

SYSTEM

*SYSTEM* ist identisch mit *QUIT* und bewirkt, daß der Interpreter das Programm beendet und ohne eine Sicherheitsabfrage zum Desktop zurückkehrt.

## TIMER

Der Zeitzähler des Atari ST wird beim Einschalten des Computers auf Null gesetzt und 200mal in der Sekunde inkrementiert. Dabei ist es unerheblich, ob zwischenzeitlich Anwendungen ausgeführt werden. Dieser Zählerstand kann direkt ausgegeben, einer Variablen übergeben oder in Bedingungsabfragen eingebunden werden.

# 4. Textoperationen

Befehle, die als Ergebnis nicht eine Zahl, sondern ein Zeichen oder Zeichenkette liefern, erkennt man an der Endung "$". Als Textoperationen werden allerdings nur die Verkettung von Zeichenketten durch den Operator "+" (z.B. *A$=B$+C$* oder *V_name$="GfA"+"-"+"BASIC"*) und die Stringfunktionen, die im folgenden Kapitel behandelt werden, bezeichnet.

Der Zeichensatz des ATARI ST gliedert sich in 3 Kategorien:

**Steuerzeichen:**
Dies sind die Zeichen mit den ASCII-Werten 0 bis 31. Durch einen Aufruf mit *PRINT CHR$(0...31)* werden diese nicht ausgegeben, sondern ihre Steuerfunktion ausgeführt. Hierzu gehören Befehle wie CARRIAGE RETURN (Wagenrücklauf: *PRINT CHR$(13)*), LINEFEED (Zeilenvorschub: *PRINT CHR$(10)*) oder BELL (Glocke: *PRINT CHR$(7)*). Zusätzlich zu den unter *INPUT* beschriebenen Ausgabemöglichkeiten von Sonderzeichen können Steuerzeichen bei Texteingabe durch gleichzeitiges Drücken von <CONTROL> und einer anderen Taste ausgegeben werden. Als Ausgabebefehl für Steuerzeichen dient in erster Linie *OUT* (siehe dort).

**Alphanumerische Zeichen:**
Buchstaben, Zahlen und Interpunktions-Zeichen mit einem ASCII-Wert von 32 bis 127. (Fast vollständig auf der Tastatur aufgedruckt).

**Sonderzeichen:**
Alle Zeichen mit einem ASCII-Wert im Bereich von 128 bis 255. Sie stellen eine Zusammenstellung von Zeichen dar, welche entweder in den unterschiedlichen Ländern oder verschiedenen wissenschaftlichen Disziplinen gebräuchlich sind (Darstellungsmöglichkeiten siehe *INPUT*).

## INSTR Zeichenkette in einem String suchen

**INSTR(Startposition,Zielstring,Suchstring)**

Übergibt einen numerischen Wert, der angibt, an welcher Position eines angegebenen Strings ein weiterer angegebener String oder Textausdruck zu finden ist. Mit *Zielstring* wird jeweils der String, bzw. die Stringvariable übergeben, die zu untersuchen ist. *Suchstring* beinhaltet den Textausdruck, bzw. die Stringvariable, nach deren Inhalt gesucht werden soll. Wird der optionale Zusatz *Startposition* nicht verwendet, wird ab dem ersten Zeichen des *Zielstrings* untersucht. Bei Verwendung von *Startposition* wird erst ab dem Zeichen untersucht, welches damit übergeben wurde. Wird *Suchstring* in *Zielstring* gefunden, erhält man die Position in *Zielstring*, ab welcher *Suchstring* steht. Kann der zu suchende String in dem zu untersuchenden String nicht gefunden werden, wird eine 0 zurückgegeben. Sind beide Strings leer ("""), wird eine 1 geliefert. Bei der Angabe von *Suchstring* ist darauf zu achten, daß zwischen Groß- und Kleinschreibung unterschieden wird. *INSTR* kann wahlweise direkt ausgegeben, einer numerischen Variablen übergeben oder in Bedingungsabfragen eingebunden werden.

## LEFT$ Linksbündige Zeichen ermitteln

**LEFT$(Zielstring,Anzahl)**

Ermittelt eine bestimmte Anzahl von Textzeichen eines vorgegebenen Strings, beginnend mit dem ersten Zeichen des Strings. Dem Befehl wird in Klammern der *Zielstring* als Variable oder Textausdruck übergeben, sowie nach einem Komma die *Anzahl* der Zeichen, welche ab dem ersten Zeichen des Zielstrings gelesen werden sollen. *Anzahl* kann auch als numerische Variable angegeben sein. Wird in der Klammer *Anzahl* nicht angegeben, wird jeweils das erste Zeichen des angegebenen Strings gelesen. Bei Angabe einer *Anzahl*, welche höher ist, als die Länge des *Zielstrings*, wird der gesamte *Zielstring* zurückgegeben. Ist der angegebene *Zielstring* ohne Inhalt ("""), wird ebenfalls

ein Nullstring ("") geliefert. *LEFT$* kann wahlweise direkt ausgegeben, einer Stringvariablen übergeben oder in einen Textausdruck eingebunden werden.

## LEN — Stringlänge ermitteln

**LEN(V_name$)**

Liefert die Länge des Strings, der in der angegeben Variablen enthalten ist. *V_name$* ist der Name einer Stringvariablen, deren Länge in Byte ermittelt werden soll. Er ist der Funktion in Klammern anzugeben. *LEN* kann direkt ausgegeben, einer Variablen übergeben oder in eine Bedingungsabfrage eingebunden werden.

## LSET [LS] — Zeichenkette linksbündig einsetzen

**LSET Ziel_var$="Stringausdruck"**

Fügt einen angegebenen String linksbündig in eine Stringvariable ein. *Stringausdruck* kann eine beliebige, in Anführungszeichen stehende Zeichenkette oder Stringvariable sein, deren Inhalt linksbündig in die vor dem Gleichheitszeichen stehende Stringvariable eingefügt wird. Dabei wird die ursprüngliche Länge von *Ziel_var$* nicht verändert. Ist die Länge von *Ziel_var$* kleiner als die Länge von *Stringausdruck*, wird der einzusetzende String bei der Länge von *Ziel_var$* abgeschnitten. Hat *Ziel_var$* dagegen eine größere Länge als der einzusetzende String, werden die restlichen Stellen des Zielstrings mit Leerzeichen aufgefüllt.

## MID$ Zeichen in Stringmitte ermitteln

**MID$(Zielstring,Start,Anzahl)**

Übergibt eine bestimmte Anzahl von Zeichen (ab einer beliebigen Startposition) aus einer Zeichenkette. Der *Zielstring*, welcher untersucht werden soll, kann direkt als Text (in Anführungszeichen) oder als Stringvariable angegeben werden. Der Parameter *Start* bestimmt, ab welchem Zeichen (z.B. 5 -> ab 5. Zeichen von links (incl.)) des Strings die angegebene *Anzahl* von Zeichen gelesen werden soll. *Start* und *Anzahl* können auch als numerische Variablen übergeben werden. Wird *Anzahl* nicht angegeben, werden alle Zeichen von *Zielstring* gelesen. Bei Angabe einer *Anzahl*, welche höher ist, als die Länge von *Zielstring* minus *Start*, werden alle nach *Start* liegenden Zeichen zurückgegeben. Ist *Zielstring* ohne Inhalt, wird ein Nullstring ("") geliefert. *MID$* kann wahlweise direkt ausgegeben, einer Stringvariablen übergeben oder in einen Textausdruck eingebunden werden.

## RIGHT$ Rechtsbündige Zeichen ermitteln

**RIGHT$(Zielstring,Anzahl)**

Ermittelt eine bestimmte Anzahl von Textzeichen eines vorgegebenen Strings, beginnend mit dem letzten Zeichen des Strings in Richtung Stringanfang. Dem Befehl wird in Klammern der *Zielstring* als Variable oder Textausdruck übergeben, sowie nach einem Komma die *Anzahl* der Zeichen, welche ab dem letzten Zeichen des Zielstrings rückwärts gelesen werden sollen. *Anzahl* kann auch als numerische Variable angegeben sein. Wird in der Klammer *Anzahl* nicht angegeben, wird jeweils das letzte Zeichen des angegebenen Strings gelesen. Bei Angabe einer *Anzahl*, welche höher ist, als die Länge des *Zielstrings*, wird der gesamte *Zielstring* zurückgegeben. Ist der angegebene *Zielstring* ohne Inhalt ("""), wird ebenfalls ein Nullstring ("") geliefert. *RIGHT$* kann wahlweise direkt ausgegeben, einer Stringvariablen übergeben oder in einen Textausdruck eingebunden werden.

## RSET [RS] Zeichenkette rechtsbündig einsetzen

**RSET Ziel_var$=Stringausdruck**

Fügt einen angegebenen String rechtsbündig in eine Stringvariable ein. *Stringausdruck* kann eine beliebige, in Anführungszeichen stehende Zeichenkette oder Stringvariable sein, deren Inhalt rechtsbündig in die vor dem Gleichheitszeichen stehende Stringvariable eingefügt wird. Dabei wird die ursprüngliche Länge von *Ziel_var$* nicht verändert. Ist die Länge von *Ziel_var$* kleiner als die Länge von *Stringausdruck*, wird der einzusetzende String linksbündig eingefügt und bei der Länge von *Ziel_var$* abgeschnitten. Hat *Ziel_var$* dagegen eine größere Länge als der einzusetzende String, werden die restlichen, links von *Stringausdruck* liegenden Stellen des Zielstrings mit Leerzeichen aufgefüllt.

## SPACE$ Zeichenkette aus Leerzeichen ausgeben

**SPACE$(Anzahl)**

Übergibt einer Ausgabeanweisung oder einem Textausdruck eine bestimmte Anzahl von Leerzeichen. Das Argument *Anzahl* bestimmt die Menge der zu übergebenden Leerzeichen. Es darf nicht negativ und nicht größer als 32767 (maximale Länge einer Stringvariablen) sein. Die mit *SPACE$* gelieferten Leerzeichen können direkt ausgegeben, an eine Variable übergeben oder in einen Textausdruck eingebunden werden.

## STRING$ Stringkette ausgeben

**STRING$(Anzahl,Stringausdruck)**
(mit Zeichen)

**STRING$(Anzahl,ASCII)**
(mit Codenummer)

Bildet eine Zeichenkette, die sich durch mehrmaliges Aneinanderfügen eines beliebigen Strings ergibt. Mit dem Parameter *Anzahl* wird bestimmt, wie oft der *Stringausdruck* mit sich selbst verkettet werden soll. *Stringausdruck* kann entweder direkt (z.B. *"Text"*) oder als Stringvariable eingesetzt werden. In einer zweiten Variante lassen sich die zu verkettenden Zeichen durch Angabe ihres ASCII-Codes einsetzen (z.B. für A , ASCII = 65). Die ASCII-Eingabe wird vom Interpreter in eine positive, ganze Zahl kleiner als 256 umgewandelt (*ASCII MOD 256*). Die entstehende Zeichenkette darf nicht mehr als 32767 Zeichen enthalten (maximale Länge einer Stringvariablen), sonst erfolgt eine Fehlermeldung.

## UPPER$ Buchstabenumwandlung klein -> groß

**UPPER$(Stringausdruck)**

Die Zeichenkette *Stringausdruck* kann entweder direkt als Text in Anführungszeichen oder als Stringvariable übergeben werden. Trifft die Funktion *UPPER$* beim Lesen dieses Strings auf einen kleingeschriebenen Buchstaben (ASCII-Werte 97 bis 129), so wandelt sie diesen in den entsprechenden Buchstaben in Großschreibweise (ASCII-Werte 65 bis 90) um. Alle anderen alphanumerischen Zeichen (bis auf einige Ausnahmen) bleiben unverändert: Die Steuerzeichen (ASCII-Werte 0 bis 31) können nicht verarbeitet werden.

# 5. Arithmetikbefehle

## 5.1 Mathematische Operationen

Die in BASIC benutzten mathematischen Befehle wurden der üblichen Schreibweise in der Mathematik nachempfunden, womit die Übersetzung von mathematischen Formeln anhand der BASIC-Kurzschreibweise recht unkompliziert ist. Bei zeitkritischen Programmteilen kann es aber sinnvoll sein, einige spezielle GfA-BASIC-Rechenbefehle wie *ADD* und *MUL* zu benutzen. Diese sind in ihrem Aufbau und Funktionsweise "Assembler-ähnlich" und sehr viel schneller als die herkömmlichen "Strichpunkt-Befehle". GfA-BASIC beherrscht folgende numerische Operatoren:

**Arithmetische Operatoren**

| | |
|---|---|
| ^ | Potenzieren |
| - | negatives Vorzeichen |
| * / | Multiplikation und Division |
| **DIV** | Ganzzahldivision (Entfernen von Dezimalstellen) |
| **MOD** | Modulo-Berechnung (Rest der Ganzzahldivision) |
| + - | Addition und Subtraktion |

**Vergleichsoperatoren**

| | | |
|---|---|---|
| = | | gleich |
| <> | | oder |
| >< | | ungleich |
| > | | größer als |
| >= | | oder |
| => | | größer oder gleich |
| < | | kleiner als |
| <= | =< | kleiner / gleich |

Abhängig vom Vergleich zweier Zahlen oder Zeichenketten erhält man den Wahrheitswert -1 (wahr) oder 0 (falsch). Der Zeichenkettenvergleich wird anhand des Größenvergleichs der ASCII-Werte der Zeichen durchgeführt. Dabei wird mit dem ersten Zeichen des Strings begonnen.

**Logische Operatoren**

Die Boolsche Algebra bedient sich der Logischen Operatoren, um den Wahrheitswert von Aussagen zu untersuchen. Hierbei sind alle Werte ungleich 0 wahr (TRUE = -1) und alle Werte gleich 0 falsch (FALSE = 0). GfA-BASIC kennt 6 logische Operatoren:

**AND** Konjunktion; das Ergebnis von AND ist wahr, wenn beide Argumente wahr sind.

```
         1 0 0 1 1 1 0 1
AND      0 1 1 1 1 0 0 1
ergibt:  0 0 0 1 1 0 0 1
```

**OR** Disjunktion; das Ergebnis von OR ist wahr, wenn eines der Argumente wahr ist.

```
         0 0 0 1 1 1 0 1
OR       0 1 1 1 1 0 0 1
ergibt:  0 1 1 1 1 1 0 1
```

**XOR** Exclusives Oder; das Ergebnis von XOR ist falsch, wenn die Argumente entweder beide wahr oder beide falsch sind.

```
         0 0 0 1 1 1 0 1
XOR      0 1 1 1 1 0 0 1
ergibt:  0 1 1 0 0 1 0 0
```

**NOT** Negation; vertauscht Wahrheitswerte ins Gegenteil.

```
NOT      1 0 0 1 1 1 0 1
ergibt:  0 1 1 0 0 0 1 0
```

**IMP** Implikation; die Folgerung IMP ist nur dann Falsch, wenn aus etwas wahrem etwas falsches folgt.

```
         0 0 0 1 1 1 0 1
IMP      0 1 1 1 1 0 0 1
ergibt:  1 0 0 1 1 1 1 1
```

**EQV** Äquivalenz; Umkehrung zu XOR. Das Ergebnis ist falsch, wenn sich die beiden Argumente unterscheiden.

```
         1 0 0 1 1 1 0 1
EQV      0 1 1 1 1 0 0 1
ergibt:  0 0 0 1 1 0 1 1
```

Mit logischen Operatoren können nicht nur Bitfolgen untersucht werden, sondern auch abhängig vom Ergebnis einer Aussage der Programmlauf (z.B. durch IF-Abfragen) gesteuert werden.

**Prioritäten**

( )
Klammern (höchste Priorität)

^
Potenzierung

-
Negatives Vorzeichen

* /
Multiplikation und Division

**DIV MOD**
Ganzzahldivision und Modulo-Berechnung

+ -
Addition und Subtraktion

= > < <> >= <=
Vergleichsoperatoren

**NOT AND OR XOR IMP EQV**
logische Operatoren (niedrigste Priorität)

Die mathematischen Operationen werden in einer festgesetzten Reihenfolge ausgeführt, die nur durch die Benutzung von Klammern aufgehoben werden kann.

## ADD [AD] Additionsbefehl

**ADD V_name1,V_name2**
**ADD V_name1,const**

Addition von Variablen mit Ergebniszuweisung. *V_name1* und *V_name2* können numerische Variablen oder numerische Feldvariablen sein. *const* ist eine numerische Konstante. Die auf das Komma folgende Komponente wird zur ersten addiert. Die Variable *V_name1* erhält das Ergebnis der Berechnung.

## DEC Dekrementierung (-1)

**DEC V_name**

Vermindert den Wert einer numerischen Variablen bzw. Feldvariablen *V_name* um den Wert 1.

## DIV Divisionsbefehl

**DIV V_name1,V_name2**
**DIV V_name1,const**

Division einer Variablen mit Ergebniszuweisung. *V_name1* und *V_name2* können numerische Variablen oder numerische Feldvariablen sein. *const* ist eine numerische Konstante. Der Wert vor dem Komma wird durch den zweiten dividiert und das Ergebnis der Rechnung anschließend *V_name1* zugewiesen.

## INC Inkrementierung (+1)

**INC V_name**

Erhöht den Wert einer numerischen Variablen bzw. Feldvariablen *V_name* um den Wert 1.

## MUL [MU] Multiplikationsbefehl

**MUL V_name1,V_name2**
**MUL V_name1,const**

Multiplikation von Variablen mit Ergebniszuweisung. *V_name1* und *V_name2* können numerische Variablen oder numerische Feldvariablen sein. *const* ist eine numerische Konstante. Der Wert nach dem Komma wird mit dem ersten multipliziert und das Ergebnis der Rechnung anschließend *V_name1* zugewiesen.

## SUB Subtraktionsbefehl

**SUB V_name1,V_name2**
**SUB V_name1,const**

Subtraktion von Variablen mit Ergebniszuweisung. *V_name1* und *V_name2* können numerische Variablen oder numerische Feldvariablen sein. *const* ist eine numerische Konstante. Die auf das Komma folgende Komponente wird von der ersten abgezogen und das Ergebnis wird der Variablen *V_name1* zugewiesen.

## SWAP Werte tauschen

**SWAP V_name1,V_name2**

Werteaustausch bei allen numerischen, alphanumerischen und boolschen Variablen bzw. Feldern, solange *V_name1* und *V_name2* vom selben Typ sind. Bei Feldern wird auch die Dimensionierung vertauscht.

## 5.2 Numerische Funktionen

Einer Funktion wird immer ein Wert in Klammern übergeben. Diesen bezeichnet man als das *Argument* der Funktion. Der wichtigste Unterschied zwischen einem Befehl und einer Funktion ist, daß diese innerhalb eines Befehls auftauchen kann (z.B. *PRINT SIN(X)* jedoch nicht *PRINT MUL A,B*). Außerdem lassen Sie sich problemlos in IF-Abfragen einbinden (z.B. *IF SQR(X)>22.351*). Alle mathematischen Funktionen des GfA-BASIC sind von der Syntax her gleich: Sie bestehen aus 3 Buchstaben, gefolgt vom Argument in Klammern. Es gibt für sie keine Abkürzung. GfA-BASIC ermöglicht es dem Programmierer auch, eigene Funktionen zu definieren (siehe *DEFFN*).

### ABS — Betrag (vorzeichenlos) ermitteln

**ABS(x)**

Gibt den absoluten Wert einer Zahl aus. Dieser ist immer größer oder gleich 0 (Betragsfunktion). Die Funktion *ABS* gibt das Argument als positiven, identischen Ausdruck ohne Vorzeichen aus.

| | |
|---|---|
| **x positiv** | **Abs(x) = x** |
| **x = 0** | **Abs(x) = 0** |
| **x negativ** | **Abs(x) = x** |

Sie findet Verwendung bei kaufmännischen Problemlösungen (Es gibt kein "Minusgeld" sondern eher Sollbeträge!) und in mathematischen Berechnungen (z.B. vor dem "Wurzelziehen" unbekannter Größen!).

## ATN — Arcustangens-Funktion

**ATN(x)**

Berechnet den Arcustangens eines Wertes (arctan = Winkel im Bogenmaß für zugehörigen Tangenswert). Das Argument X ist ein Tangenswert, von dem der Winkel in "Radian" zurückgerechnet wird. Die Funktion *ATN(X)* berechnet einen Wert zwischen -Pi/2 und +Pi/2 . Benötigt man die Winkelangabe in Grad, so muß das Ergebnis mit 180/Pi multipliziert werden.

## COS — Cosinus-Funktion

**COS(x)**

Berechnet den Cosinus eines Winkels im Bogenmaß. Der Cosinus eines Winkels ist in rechtwinkligen Dreiecken geometrisch als Teilungsergebnis seiner Ankathete durch die Hypothenuse definiert. Das Argument X ist ein Winkel in "Radian", für den der Cosinuswert berechnet wird. Soll der Winkel in Grad eingegeben werden, muß der Wert vorher mit Pi/180 multipliziert werden.

## EXP — Exponential-Funktion

**EXP(x)**

Berechnet den Wert der Exponential-Funktion, dies ist die Konstante e, erhöht um einen Exponenten. Die Exponentialfunktion gibt das Argument zur Basis e aus, dabei kann der Exponent *X* eine beliebige Zahl sein. Die Konstante e heißt Euler'sche Zahl (e = 2.718281828...) und ist die Basis aller natürlichen Logarithmen.

## FIX — Ganzzahlfunktion

**FIX(x)**

Übergibt den ganzzahligen Anteil (Integer) einer Realzahl, nach Abschneiden der Nachkommastellen. *X* ist ein beliebiger numerischer Ausdruck. Die Funktion rundet Zahlen weder auf noch ab, sondern entfernt nur die Dezimalstellen. Der Nachkommaanteil wird durch *FRAC* geliefert. *FIX* ist identisch mit *TRUNC*.

## FRAC — Dezimalstellen-Funktion

**FRAC(x)**

Liefert den Nachkommaanteil einer Realzahl. Das Argument *X* kann ein beliebiger numerischer Ausdruck sein. Die Funktion übergibt den Dezimalanteil, falls *X* eine reelle Zahl ist, und den Wert 0, falls *X* eine Integerzahl ist. *FRAC* ist komplementär zu *FIX*.

## INT — Ganzzahlfunktion (Integer)

**INT(x)**

Wandelt eine Realzahl in eine Integerzahl um. Es wird die nächstkleinere Ganzzahl zurückgegeben. Das Argument *X* ist eine beliebige positive oder negative Zahl, deren Nachkommastellen ignoriert werden. Das Ergebnis ist immer eine Integerzahl.

## LOG,LOG10 Logarithmus-Funktion

**LOG(x)**
(natürlicher Logarithmus)

**LOG10(x)**
(dekadischer Logarithmus)

Berechnet den natürlichen bzw. den dekadischen Logarithmus einer Zahl. Der Logarithmus der Zahl *X* ist der Exponent der Basis (hier e oder 10). Diese Potenzierung ergäbe die Zahl *X*. Dabei ist e die Eulersche Zahl. Der numerische Ausdruck *X* muß in jedem Fall größer 0 sein!

## MAX Größten Ausdruck einer Liste ausgeben

**MAX(Ausdruck1,Ausdruck2,Ausdruck3...)**

Gibt den zahlenmäßig größten Wert eines Ausdrucks oder den durch ">" ermittelten größten String zurück. *Ausdruck* kann eine beliebiger numerischer- oder Textausdruck, Wert oder String, bzw. Variable sein. Alle Ausdrücke müssen demselben Typ angehören. Bei numerischen Vergleichen wird die größte Zahl übergeben. Bei Stringvergleichen wird der größte String ermittelt, indem der Reihe nach alle Einzelzeichen der zu vergleichenden Strings überprüft werden. Haben beide Zeichen denselben ASCII-Wert, werden solange die nächsten beiden Zeichen geprüft, bis Sie sich unterscheiden oder einer der beiden Strings keine Zeichen mehr enthält. Im ersten Fall ist der Ausdruck größer, dessen zuletzt geprüftes Zeichen den größeren ASCII-Wert besitzt. Im zweiten Falle ist es der String mit der größeren Länge.

## MIN Kleinsten Ausdruck einer Liste ausgeben

**MIN(Ausdruck1,Ausdruck2,Ausdruck3...)**

Gibt den kleinsten Wert eines Ausdrucks oder den durch "<" ermittelten kleinsten String zurück. *Ausdruck* kann ein beliebiger numerischer- oder Textausdruck, Wert oder String bzw. Variable sein. Alle Ausdrücke müssen demselben Typ angehören. Bei numerischen Vergleichen wird die kleinste Zahl übergeben. Bei Stringvergleichen wird der kleinste String ermittelt, indem der Reihe nach alle Einzelzeichen der zu vergleichenden Strings überprüft werden. Haben beide Zeichen denselben ASCII-Wert, werden solange die nächsten beiden Zeichen geprüft, bis Sie sich unterscheiden oder einer der beiden Strings keine Zeichen mehr enthält. Im ersten Fall ist der Ausdruck kleiner, dessen zuletzt geprüftes Zeichen den kleineren ASCII-Wert besitzt. Im zweiten Falle ist es der String mit der geringeren Länge.

## PI Kreiszahl

**PI**

Übergibt den Wert der Kreiszahl Pi. Die Kreiszahl ist eine geometrische Konstante und beschreibt das Verhältnis zwischen Durchmesser und Umfang eines Kreises. Pi = 3.141592653...

Der Wert *PI* ist unersätzlich, wenn es darum geht, auf Kreise zurückzuführende geometrische Formen zu beschreiben oder ihre Flächen und Volumen zu berechnen.

## RANDOM — Integer-Zufallszahl

**RANDOM(n)**

Übergibt eine ganzzahlige Zufallszahl aus einem beliebigen Integer-Zahlenbereich. Die Grenze des Zahlenbereichs *N* darf eine beliebige Zahl sein. Es wird eine Integerzahl aus dem Zahlenbereich zwischen 0 (inclusiv) und *N* (exclusiv) ausgegeben, wobei *N* auch negativ sein kann.

## RND — Dezimalstellen-Zufallszahl (0 < x < 1)

**RND(Dummy)**
**RND**

Übergibt eine Zufallszahl zwischen 1 und 0. Das Scheinargument *Dummy* wird nicht beachtet, es wird ein Zufallswert mit 11 Nachkommastellen generiert, der im Bereich zwischen 0 (inclusiv) und 1 (exclusiv) liegt. z.B. 0.63859672569

## SGN — Vorzeichen ermitteln

**SGN(x)**

Gibt das Vorzeichen einer Zahl aus. Der numerische Ausdruck *X* wird auf sein Vorzeichen hin untersucht. Die Signum-Funktion übergibt:

| | | | |
|---|---|---|---|
| +1 | wenn | x>0 | (Positiv) |
| -1 | wenn | x<0 | (Negativ) |
| 0 | wenn | x=0 | (Null) |

## SIN — Sinus-Funktion

**SIN(x)**

Berechnet den Sinus eines Winkels im Bogenmaß. Der Sinus eines Winkels ist geometrisch als Teilungsergebnis von Gegenkathete durch Hypothenuse im rechtwinkligen Dreieck definiert. Das Argument *X* ist ein Winkel in "Radian", für den der Sinuswert berechnet werden soll. Soll der Winkel in Grad eingegeben werden, muß dieser mit Pi/180 multipliziert werden.

## SQR — Wurzelfunktion

**SQR(x)**

Berechnet die quadratische Wurzel einer Zahl. Das Argument *X*, auch "Radikand" genannt, ist ein numerischer Ausdruck, der nicht negativ sein darf. Die Funktion übergibt die 2.Wurzel, wird eine höhere Wurzel gebraucht, so muß diese über den Umweg der Potenzierung mit gebrochenem Exponenten berechnet werden: x^1/3 , x^1/4 ...

## TAN — Tangens-Funktion

**TAN(x)**

Berechnet den Tangens eines Winkels im Bogenmaß. Der Tangens eines Winkels ist geometrisch als Teilungsergebnis von Gegenkathete durch Ankathete eines rechtwinkligen Dreiecks definiert. Das Argument *X* ist ein Winkel in "Radian", für den der Tangenswert berechnet werden soll. Soll der Winkel in Grad eingegeben werden, muß dieser mit Pi/180 multipliziert werden. Das Gegenstück ist die Funktion *ATN*.

TRUNC(x)

Übergibt den ganzzahligen Anteil (Integer) einer Realzahl, nach Abschneiden der Nachkommastellen. *X* ist ein beliebiger numerischer Ausdruck. Die Funktion rundet Zahlen weder auf noch ab, sondern entfernt nur die Dezimalstellen. Diesen Anteil nach dem Komma kann die FRAC-Funktion liefern.

# 6. Grafik

## 6.1 Grafikdefinition

Mit den folgenden Befehlen lassen sich die Modi festlegen, in denen Grafikbefehle ausgeführt werden sollen. Es handelt sich hauptsächlich um Vereinbarungen über die Farbe, Strichstärke, Linienend- und anfangsform, Musterart, Grafikmodus usw. Diese Grafikparameter unterliegen bei Programmstart einer Grundeinstellung, z.B. beträgt die Linienbreite für den LINE-Befehl 1 Pixel, wenn vorher im Programm noch kein *DEFLINE* verwendet wurde, das die Linienbreite verändert. Zu beachten ist, daß die meisten Grafikdefinitionen nach Ausführung des entsprechenden Grafikbefehls wieder in ihren Grundzustand zurückgesetzt werden sollten, weil auch andere Befehle auf diese Definitionen zurückgreifen. Zum Beispiel verändert der oben genannte DEFLINE-Befehl auch die Linienmodi für die BOX-, CIRCLE- oder ELLIPSE-Befehle.

### COLOR [CO] Linienfarbe bestimmen

**COLOR Farbe**

Bestimmt die Farbe für linien- und punktezeichnende Grafikbefehle. Jede Auflösungsstufe bietet eine bestimmte Anzahl von Farben, mit denen gezeichnet werden kann. Die aktuelle Farbeinstellung für die Befehle *BOX*, *RBOX*, *CIRCLE*, *DRAW*, *ELLIPSE*, *LINE* und *PLOT* wird mit dem Parameter *Farbe* in der COLOR-Anweisung ausgewählt:

| | |
|---|---|
| Hohe Auflösung | *Farbe* = 0 oder 1 |
| Mittlere Auflösung | *Farbe* = 0,1,2 oder 3 |
| Niedrige Auflösung | *Farbe* = 0,1,2...15 |

Bei Farbbetrieb können die Farbwerte der einzelnen durch *COLOR* ansprechbaren Farbregister mit dem SETCOLOR-Befehl definiert werden.

Mit Muster füllende Grafikbefehle wie *PBOX*, *PRBOX*, *PCIRCLE* und *PELLIPSE* können nicht durch *COLOR* beeinflusst werden (Hierzu siehe unter *DEFFILL*).

## DEFFILL [DEFF] Füllmuster bestimmen

**DEFFILL Farbe,Füllstil,Muster**
(Muster wählen)

**DEFFILL Farbe,F_muster$**
(Muster definieren)

Bestimmt das zu benutzende Füllmuster, die Füllfarbe und erlaubt es, eigene Muster zu definieren. Die mit Mustern füllenden Grafikbefehle wie *FILL*, *PBOX*, *PCIRCLE*, *PELLIPSE* und *PRBOX* benötigen die Angabe eines aktuellen Füllmusters. Mit den Parametern *Füllstil* und *Muster* kann eines der 36 vom Betriebssystem zur Verfügung gestellten Füllmuster ausgewählt werden:

| Füllstil | Muster |
|---|---|
| 0 = Hintergrundfarbe (Voll) | entfällt |
| 1 = Objektfarbe (Voll) | entfällt |
| 2 = Punktiert | 1 bis 24 |
| 3 = Schraffiert | 1 bis 12 |
| 4 = Selbstdefiniert | "ATARI" oder Eigenes |

Mit der Variante *DEFFILL Farbe,F.muster$* läßt sich ein eigenes Füllmuster einrichten. Ähnlich wie bei der Mausformdefinition wird hier ein 32 Byte langer String in die Stringvariable *F_muster* eingesetzt. Das Füllmuster besteht aus 16*16 Pixel. Es brauchen hier nur 16 Words übergeben werden, die der Reihe nach die Bitmuster der einzelnen Füllmusterzeilen beinhalten. Dies geschieht am besten über die Funktion *MKI$*.

```
Restore F.illdatas
For I%=1 To 16
  Read F.zeile%
  F.muster$=F.muster$+Mki$(F.zeile%)
Next I%
Deffill ,1,1
Pbox 100,100,320,200
Deffill 1,0
Pbox 320,100,540,200
Deffill 1,4
Pbox 100,200,320,300
Deffill 1,F.muster$
Pbox 320,200,540,300
For I=1 To 25
  Deffill ,2,I
  Pbox I*23,0,I*23+23,40
  Deffill ,3,I Mod 13
  Pbox I*23,360,I*23+23,399
Next I
U=Inp(2)
Edit
F.illdatas:
Data &X0000000000111000
Data &X0000000001010000
Data &X0001111110111000
Data &X0010011101000100
Data &X0100001110000010
Data &X1010000000000010
Data &X1100000000000001
Data &X1010000000000001
Data &X1101000000000001
Data &X1010100000001011
Data &X0101010101010100
Data &X0010101010101000
Data &X0001010101010000
Data &X0000111010110000
Data &X0000000000000000
Data &X0000000000000000
```

**DEFLINE Linienstil,Liniendicke,Anfangsform,Endform**

Bestimmt die Darstellungsart der Linien, die mit den Befehlen *BOX*, *CIRCLE*, *LINE*, *RBOX*, *ELLIPSE* und *DRAW* sichtbar gemacht werden können. Es können vier Parameter übergeben werden:

**Linienstil**

```
            0 = Linie in Hintergrundfarbe
            1 = durchgezogene Linie
            2 = gestrichelte Linie (kleine Abstände)
            3 = gepunktete Linie
            4 = Punkt-Strich-Linie
            5 = gestrichelte Linie (große Abstände)
            6 = Strich-Punkt-Punkt-Linie
-1 bis -32767 = selbstdefinierter Linienstil
```

Der selbstdefinierte Linienstil setzt sich aus einem Word (16 Bit) zusammen, wobei jedes gesetzte Bit einen Punkt in der Linie darstellt und ein nicht gesetztes Bit eine freie Stelle. Diese Zahl muß als Minuswert im ersten Parameter eingesetzt werden, um einen Linienstil mit dem angegebenen Bitmuster zu erzeugen. Die Linie setzt sich dann aus dem Vielfachen dieser 16 Bit zusammen. Eine Darstellung der verschiedenen Stile ist jedoch nur möglich, wenn mit Liniendicke 1 gearbeitet wird

**Liniendicke**
Der zweite Parameter legt die Liniendicke fest. Es ist eine maximale Liniendicke von 40 Pixel möglich. Dabei ist zu beachten, daß sich die Dicke in Zweierschritten erhöht.

**Anfangsform/Endform**
Die Formen des Linienanfangs und -endes lassen sich mit den letzten beiden Parametern bestimmen:

| | |
|---|---|
| 1 = | eckig |
| 2 = | pfeilförmig |
| 3 = | rund |

## DEFMARK [DEFM] Markierungssymbol bestimmen

**DEFMARK Markerfarbe,Markertyp,Markergröße**

Es können Farbe, Typ und Größe der Markierungssymbole bestimmt werden. Als Marker stehen 6 Typen zur Auswahl:

Typ 1 = ein Ein-Pixel-Punkt
Typ 2 = ein Plus-Zeichen
Typ 3 = ein sechszackiger Stern
Typ 4 = ein liegendes Rechteck
Typ 5 = ein Diagonalkreuz bzw. X
Typ 6 = eine liegende Raute

Alle Marker besitzen unveränderlich eine Liniendicke von einem Pixel. Die Größe ist nur in einer Schrittweite von 20 veränderbar.

## DEFMOUSE [DEFMO] Mausform bestimmen

**DEFMOUSE Form**
**DEFMOUSE Maus$**

Aufruf von selbstdefinierten oder systemeigenen Mausformen.

Mausformen (vordefiniert):

Form 0 = Pfeil
Form 1 = Doppelklammer
Form 2 = Biene
Form 3 = zeigende Hand
Form 4 = offene Hand
Form 5 = Fadenkreuz fein
Form 6 = Fadenkreuz grob
Form 7 = Fadenkreuz weiß mit Rand

Wird statt *Form* eine Stringvariable angegeben, kann mit deren Inhalt eine Mausform frei definiert werden. In diesem Fall sind alle Daten als 2-Byte-Werte im MKI$-Format zu übergeben. Das Mausraster selbst besteht aus 16 Zeilen und 16 Spalten.

Word 1 X-Koordinate des Aktionspunktes innerhalb der Mausform

Word 2 Y-Koordinate des Aktionspunktes innerhalb der Mausform

Auf den Aktionspunkt werden anschließend alle Mausaktionen (z.B. *MOUSEX*, *MOUSEY*) bezogen.

```
Maus$=Maus$+Mki$(1)+Mki$(1)          = links oben
Maus$=Maus$+Mki$(1)+Mki$(65535)      = links unten
Maus$=Maus$+Mki$(65535)+Mki$(65535)  = rechts unten
Maus$=Maus$+Mki$(65535)+Mki$(1)      = rechts oben
```

Word 3 immer *MKI$(1)*

Word 4 Maskenfarbe (Hintergrund des Mausbildes) (weiß = *MKI$(0)* / schwarz = *MKI$(1)*)

Word 5 Cursorfarbe (Mausbild)

Word 6 bis Word 21
Bitmuster der Mausmaske

Word 22 bis Word 37
Bitmuster des Mausbildes

```
Restore Mausdatas.maske
For I%=0 To 15
  Read D.atm%
  Mausmaske$=Mausmaske$+Mki$(D.atm%)
Next I%
Restore Mausdatas.form
For I%=0 To 15
  Read D.atf%
  Mausform$=Mausform$+Mki$(D.atf%)
Next I%
M_ausdaten$=Mki$(1)+Mki$(65535)+Mki$(1)+Mki$(0)+Mki$(1)
M_ausdaten$=M_ausdaten$+Mausmaske$+Mausform$
Defmouse M_ausdaten$
Deffill ,2,8
Pbox 100,100,540,300
Repeat
  Mouse X,Y,K
  Plot X,Y
Until Mousek
Edit
Mausdatas.maske:
Data 112,248,508,1022,2047,4095,8191,8190,16380
Data 16376,32752,32736,65472,65280,64512,61440
Mausdatas.form:
Data 32,80,168,332,658,1317,2634,3220,5416
Data 4688,8608,8384,21248,19456,61440,49152
```

## DEFTEXT [DEFT] Grafik-Text-Modi bestimmen

**DEFTEXT Textfarbe,Textart,Textwinkel,Textgröße**

Es können die verschiedenen Darstellungsarten der ausgegebenen Schriftzeichen bestimmt werden. Außerhalb von GEM-Fenstern hat dieser Befehl nur Auswirkung auf die mit *TEXT* ausgegebenen Zeichen. Innerhalb von Fenstern (*Handle* 0 bis 5) können auch mit *PRINT* ausgegebene Zeichen verändert werden.

Die Parameter haben folgende Bedeutung:

**Farbe** Bestimmung des Registers, aus welchem der Text seine Farbe beziehen soll.

**Typ** Einstellung der Textart. Es stehen 31 Arten zur Verfügung:

```
Wert 0  = normal
Wert 1  = bold
Wert 2  = normal lightend
Wert 3  = bold lightend
Wert 5  = italic
Wert 8  = bold underlined
Wert 16 = outlined 1
Wert 17 = outlined 2
Wert 20 = italic outlined usw.
```

**Rotation** Einstellung des Textrotationswinkels.

```
Wert 0    = normale Lage.
Wert 900  = Text wird senkrecht von unten nach oben
            ausgegeben.
Wert 1800 = Text wird auf dem Kopf stehend
            waagerecht von rechts nach links
            ausgegeben
Wert 2700 = Text erscheint senkrecht von oben nach
            unten
```

**Größe** Einstellung der Texthöhe (Werte von 0 bis 26):

```
Wert 0  = nichts
Wert 4  = Desktop-Icon-Schrift
Wert 6  = Farbschrift (8*8)
Wert 13 = Normalschrift (8*16)
Wert 26 = Mammutschrift
```

**GRAPHMODE Modus**

Für alle Grafikausgaben, außer der Rasteroperation *PUT*, gelten vier verschiedene Grafikmodi:

**Modus = 1 (Replace)**
Das verwendete Grafikelement (*PBOX*, *LINE* etc.) wird vollflächig dargestellt. Alles, was sich darunter befindet, wird davon überdeckt und ersetzt (neuer Punkt = Farbmaske des neuen Punktes AND neuer Punkt).

**Modus = 2 (Transparent)**
Es werden nur dort Bildpunkte gesetzt, wo dem neuen Grafikelement Farben zugeordnet wurden. Wo keine Farben darzustellen sind, bleibt der alte Hintergrund erhalten. Die neue Fläche erscheint also durchsichtig (neuer Punkt = (Farbmaske des neuen Punktes AND neuer Punkt) OR (Farbe des alten Punktes AND NOT neuer Punkt)).

**Modus = 3 XOR (Exklusiv Oder)**
Es werden nur dort Punkte gesetzt, wo vorher noch keine vorhanden waren. Werden zwei Grafikelemente der gleichen Art in diesem Modus auf exakt dieselbe Stelle gesetzt, wird das erste vom zweiten wieder gelöscht, ohne den vorherigen Hintergrund zu zerstören (neuer Punkt = neuer Punkt XOR Farbe des alten Punktes).

**Modus = 4 (Revers transparent)**
Dieser Modus ist identisch mit Modus 2. Der Unterschied ist, daß die zu zeichnenden Farben revers dargestellt werden. Werden also zwei dieselben Grafikelemente einmal im Modus 2 und einmal im Modus 4 dargestellt, so bildet das zweite das Negativ des ersten (neuer Punkt = (Farbe des alten Punktes AND neuer Punkt) OR (Farbmaske des neuen Punktes AND NOT neuer Punkt)).

**SETCOLOR Register,Rotanteil,Grünanteil,Blauanteil**
**SETCOLOR Register,Mischwert**

Mit diesem Befehl können die Farbtöne (RGB-Mischung) der einzelnen Farbregister bestimmt werden. Es kann entweder die Farbe des jeweiligen Farbregisters unter Angabe der drei Farbanteile (RGB = 0 bis 7) oder mit einem Gesamtmischwert (1 bis 1911) definiert werden. Im zweiten Fall setzt sich der Wert wie folgt zusammen:

```
  Rotanteil  * 256
+ Grünanteil * 16
+ Blauanteil
-----------------
= "Mischwert"
```

Farbregister mit spezieller Bedeutung:

Register 0 = Hintergrundfarbe
Register 1 = Print-Ausgabetextfarbe
Register 3 = Textfarbe des BASIC-Editors

## 6.2 Grafikbefehle

Allen Grafikbefehlen werden Koordinaten übergeben, auf die sich die Ausgabe bezieht. Diese Koordinaten sind relativ zu den jeweiligen Bildschirm-Nullpunkten zu sehen. Wurden keine GEM-Fenster geöffnet, ist dies die linke obere Bildschirmecke. Breitenangaben werden nachfolgend als X-Werte, Höhenangaben als Y-Werte dargestellt. Die Koordinatenangaben können auch außerhalb des sichtbaren Bildschirmbereichs liegen. Es wird dann jedoch die verwendete Figur nur bis zu den Bildschirm-, bzw. Fensterrändern gezeichnet.

Treten im folgenden Grafikbefehle mit vorangestelltem "P" auf (z.B. *BOX - PBOX*), so bedeutet dies, daß der Befehl die gleiche Wirkung hat, mit der Ausnahme, daß er die gezeichnete Fläche zusätzlich mit dem voreingestellten Füllmuster ausfüllt.

### BOX, PBOX [B,PB] Rechteck zeichnen

**BOX X_links,Y_oben,X_rechts,Y_unten**
(leeres Rechteck)

**PBOX X_links,Y_oben,X_rechts,Y_unten**
(gefülltes Rechteck)

Zeichnet ein Rechteck, leer oder mit Muster gefüllt. Dem Befehl werden 2 Koordinatenpaare (*X_links/Y_oben* und *X_rechts/Y_unten*) übergeben. Sie bezeichnen die beiden diagonal gegenüber liegenden Ecken des Rechtecks.

## CIRCLE, PCIRCLE [C, PC] Kreis zeichnen

**CIRCLE X_pos,Y_pos,Radius**
(Vollkreis)

**PCIRCLE X_pos,Y_pos,Radius**
(Vollkreis gefüllt)

**CIRCLE X_pos,Y_pos,Radius,Alpha,Beta**
(Kreisbogen)

**PCIRCLE X_pos,Y_pos,Radius,Alpha,Beta**
(Kreisbogen gefüllt)

Zeichnet einen Kreis oder einen Kreisausschnitt, leer oder gefüllt. Das Koordinatenpaar *X_pos/Y_pos* bestimmt den Mittelpunkt des Kreises. Mit *Radius* wird der Abstand des Kreisumfangs zu seinem Mittelpunkt bezeichnet. Werden Anfangswinkel *Alpha* und Endwinkel *Beta* an die Parameterliste angehängt, so wird ein Kreisbogen ("Tortenstück") gezeichnet. Ist *Alpha* z.B = 0, so fängt der Kreisbogen rechts auf der X-Achse (durch den Mittelpunkt) des Kreises an. Der Endwinkel *Beta* wird von hier ausgehend entgegen der Uhrzeigerrichtung abgeschritten und bezeichnet den Endpunkt des Kreisbogens. Beide Winkelangaben erfolgen in 1/10-tel Gradschritten (z.B. 900 entspricht 90 Grad).

## DRAW [DR] Punkte zeichnen und verbinden

**DRAW X_pos,Y_pos**
(1 Punkt zeichnen)

**DRAW TO X_pos,Y_pos**
(1 Punkt anbinden)

**DRAW X1,Y1 TO X2,Y2...TO Xn,Yn**
(Punkte verbinden)

Zeichnet einzelne Grafikpunkte und verbindet diese gegebenenfalls durch Linien. Mit *DRAW* wird durch Angabe des Koordinatenpaars *X_pos/Y_pos* je ein Grafikpunkt gezeichnet. In diesem Fall ist *DRAW* identisch mit dem Befehl *PLOT*. Der optionale Zusatz *TO* bewirkt, daß der angegebene Punkt zusätzlich durch eine gerade Linie mit dem zuletzt gezeichneten Grafikpunkt (kann z.B. auch der Endpunkt eines LINE-Befehls sein) verbunden wird. Wird der Zusatz *TO* mehrmals verwendet, kann eine geschlossene Punktekette gezeichnet werden.

## ELLIPSE, PELLIPSE [ELL,PE] Ellipse zeichnen

**ELLIPSE X_pos,Y_pos,A,B**
(Ellipse)

**PELLIPSE X_pos,Y_pos,A,B**
(Ellipse gefüllt)

**ELLIPSE X_pos,Y_pos,A,B,Alpha,Beta**
(Ellipsenbogen)

**PELLIPSE X_pos,Y_pos,A,B,Alpha,Beta**
(Ellipsenbogen gefüllt)

Zeichnet eine Ellipse oder einen Ellipsenbogen. Eine Ellipse wird durch ihre beiden Halbachsen *A* und *B* definiert. *A* ist der Ellipsenradius in X-Richtung und *B* der in Y-Richtung. Der Mittelpunkt der Ellipse wird durch das Koordinatenpaar *X_pos/Y_pos* festgelegt. Werden die

Werte *Alpha* und *Beta* an die Parameterliste angehängt, dann wird ein Ellipsenbogen (Ausschnitt) vom Anfangswinkel *Alpha* bis zum Endwinkel *Beta* gezeichnet. Diese Winkelangaben erfolgen in 1/10 tel Grad-Schritten (900 entspricht z.B. 90 Grad). Ist ein Winkel = 0 , so zeigt er in Richtung der positiven X-Achse.

## FILL [FI] Flächen mit Muster füllen

**FILL X_pos,Y_pos**

Füllt eine umrandete Fläche mit einem Füllmuster. Das Koordinatenpaar *X_pos/Y_pos* gibt die Lage des Bildschirmpunktes an, bei dem angefangen wird, die Fläche zu füllen. Zur Wahl des Füllmusters siehe unter *DEFFILL*.

## LINE [LI] Linie zeichnen

**LINE X1,Y1,X2,Y2**

Verbindet zwei Bildschirmpunkte mit einer Linie. Es werden die beiden Koordinatenpaare *X1/Y1* und *X2/Y2* miteinander durch eine gerade Linie verbunden. Zur Wahl der Linien-Eigenschaften siehe unter *DEFLINE*.

## PLOT [PL] Punkt zeichnen

**PLOT X_pos,Y_pos**

Zeichnet einen Grafikpunkt. Das Koordinatenpaar *X_pos/Y_pos* bestimmt die Lage des Bildschirmpunktes, der gesetzt werden soll. *PLOT* wird von der Linienbreite-Definition durch *DEFLINE* nicht beeinflußt, bis auf die Ausnahme, daß Linienanfangs und -endform als rund definiert wurden.

## POINT Farbwert (für Bildschirmpunkt) ermitteln

**POINT X_pos,Y_pos**

Untersucht einen Bildschirmpunkt auf seinen Farbwert. *X_pos/Y_pos* ist das Koordinatenpaar des Bildschirmpunktes, der untersucht werden soll. Die Funktion übergibt bei

| | |
|---|---|
| niedriger Auflösung: | 0 bis 15 |
| mittlerer Auflösung: | 0, 1, 2 oder 3 |
| hoher Auflösung: | 0 oder 1 |

Diese Werte entsprechen den Parametern bei *COLOR*. Der gelieferte Wert kann direkt ausgegeben, einer Variablen übergeben oder in eine Bedingungsabfrage eingebunden werden.

## POLYLINE [POL] Polygon zeichnen

**POLYLINE Pkte,Xp(),Yp()**
**POLYLINE Pkte,Xp(),Yp() OFFSET Xdiff,Ydiff**

Ermöglicht es, einen beliebigen Linienzug zu zeichnen. Der Parameter *Pkte* bestimmt, wieviele Punkte durch Linien verbunden werden sollen (max. 128 Punkte). Die Felder *X_pos()* und *Y_pos()* enthalten jeweils die X- und Y-Koordinaten der Eckpunkte in gleicher Reihenfolge. Dabei ist das erste Element des Feldes (*X_pos(0)* bzw. *Y_pos(0)*) jeweils die entsprechende Koordinate des ersten Eckpunktes. Soll ein geschlossenes Vieleck gebildet werden, dann muß ein zusätzlicher Eckpunkt, nämlich der Anfangspunkt an die Liste angehängt werden (Endpunkt = Anfangspunkt). Somit würde z.B. der Parameter *Pkte* für ein Sechseck den Wert 7 enthalten.

Durch Anhängen der OFFSET-Ergänzung ist es möglich, den Linienzug unter Angabe einer Differenz zu den Punkte-Koordinaten (*Xdiff/Ydiff*) in die entsprechende Richtung zu verschieben.

Zum Füllen von Polygonflächen und zur Eckpunkt-Markierung von Linienzügen siehe unter *POLYFILL* und *POLYMARK*.

## POLYFILL [POLYF] Polygon zeichnen, gefüllt

**POLYFILL Pkte,Xp(),Yp()**
**POLYFILL Pkte,Xp(),Yp() OFFSET Xdiff,Ydiff**

Es wird ein beliebiger Linienzug gezeichnet, die darin eingeschlossenen Flächen werden gefüllt. Es gelten die gleichen Ausführungen wie zu *POLYLINE*, nur daß zusätzlich die zwischen den einzelnen Linien liegenden Flächen mit dem eingestellten Füllmuster ausgefüllt werden. Da der Füllprozeß nach Zeichen der Linien einsetzt, sind die gefüllten Flächen bei sich überschneidenden Linien nicht vorhersehbar.

## POLYMARK [POLYM] Polygon-Eckpunkte

**POLYMARK Pkte,Xp(),Yp()**
**POLYMARK Pkte,Xp(),Yp() OFFSET Xdiff,Ydiff**

Es werden die mit *DEFMARK* vorbestimmten Markersymbole auf die im POLYLINE-Array definierten Eckpunkte gesetzt. Es gelten die gleichen Ausführungen wie zu *POLYLINE*, nur daß anstatt eines Linienzuges das eingestellte Markierungssymbol auf die definierten Eckpunkte gezeichnet wird.

## RBOX, PRBOX [RB, PRB] Rechteck abgerundet

**RBOX X_links,Y_oben,X_rechts,Y_unten**
(leeres Rechteck)

**PRBOX X_links,Y_oben,X_rechts,Y_unten**
(gefülltes Rechteck)

Zeichnet ein Rechteck mit runden Ecken, leer oder mit Muster gefüllt. Dem Befehl werden 2 Koordinatenpaare (*X_links/Y_oben* und *X_rechts/Y_unten*) übergeben. Sie bezeichnen die beiden diagonal gegenüber liegenden Ecken des Rechtecks.

## SPRITE [SPR] Sprite setzen und löschen

**SPRITE Defstring$,x,y**
(Sprite plazieren)

**SPRITE Defstring$**
(Sprite löschen)

Es können Software-Sprites generiert, auf dem Bildschirm plaziert oder gelöscht werden. Zur Sprite-Definition sind ebenso wie bei *DEFMOUSE* alle Daten als 2-Byte-Werte im MKI$-Format einer Stringvariablen zu übergeben. Das Spriteraster besteht ebenfalls aus 16 Zeilen und 16 Spalten.

Word 1 X-Koordinate des Aktionspunktes innerhalb der Mausform

Word 2 Y-Koordinate des Aktionspunktes innerhalb der Mausform

```
S.pr$=S.pr$+Mki$(1)+Mki$(1)      = links oben
S.pr$=S.pr$+Mki$(1)+Mki$(16)     = links unten
S.pr$=S.pr$+Mki$(16)+Mki$(16)    = rechts unten
S.pr$=S.pr$+Mki$(16)+Mki$(1)     = rechts oben
```

Word 3 Modus (normal = *MKI$(0 / XOR=MKI$(1))*

Word 4 Maskenfarbe (Hintergrund des Spritebildes) (weiß = *MKI$(0) / SCHWARZ=MKI$(1)*)

Word 5 Cursorfarbe (Spritebild)

Word 6 bis Word 37
beginnend mit dem Bitmuster der Spritemaske abwechselnd die Zeilen der Maske und die Zeilen der Form.

Unter Angabe des so definierten Spritestrings und der Koordinate, an welcher der Sprite erscheinen soll, wird der Sprite auf dem Bildschirm plaziert. Bei jedem Aufruf desselben Sprites mit veränderten Koordinaten wird er an der vorherigen Position gelöscht. Soll der Sprite vollständig vom Bildschirm entfernt werden, ist die Angabe der Koordinate zu unterlassen.

## TEXT [T] Text im Grafikmodus ausgeben

**TEXT x,y,"Text"**
(Text)

**TEXT x,y,Stringausdruck**
(Text zusammengesetzt)

**TEXT x,y,V_name$**
(als Stringvariable)

**TEXT x,y,Länge,V_name$**
(variabler Zeichenabstand)

**TEXT x,y,-Länge,V_name$**
(variabler Wortabstand)

Es wird Grafiktext an einer beliebigen Stelle des Bildschirms ausgeben. Der Ausgabetext kann sowohl direkt (*"Text"*), als Stringvariable *V_name$*, sowie auch als zusammengesetzter Text (z.B. *Stringausdruck = A$+STR$("1")*) angegeben werden. Der Text kann pixelgenau (*x,y*) gesetzt werden. Die Textattribute werden mit dem Befehl *DEFTEXT* festgelegt. *X,Y* steht für das Koor-

dinatenpaar des Punktes, an den der Text linksbündig angelegt wird. Der Zusatz *Länge* ist optional (wahlfrei). Wird eine *Länge* für die Textausgabe angegeben, so wird der Text in seiner optischen Länge der Vorgabe angepaßt. Dies geschieht durch Streckung oder Stauchung der Buchstabenabstände (wenn *Länge* positiv) oder der Wortabstände (wenn *Länge* negativ).

## 6.3 Bildschirmoperationen

### GET — Bildschirmbereich speichern

**GET X1,Y1,X2,Y2,V_name$**

Liest einen Bereich des Bildschirms in eine Stringvariable ein. Mit den Koordinaten des oberen, linken Eckpunktes (*X1,Y1*) und denen des unteren, rechten Eckpunktes (*X2,Y2*) kann ein Bildschirmausschnitt definiert werden, welcher als Bitmuster in eine Stringvariable eingelesen wird. Dieses läßt sich mit dem Befehl *PUT* an einer beliebigen Stelle wieder auf dem Bildschirm ausgeben.

### PUT — Bildschirmbereich setzen

**PUT x,y,V_name$,modus**
**PUT x,y,V_name$**

Wenn ein Bildschirmbereich vorher mit *GET* in eine Stringvariable als Bitmuster abgelegt wurde, zeichnet *PUT* diesen Ausschnitt an eine beliebige Stelle mit den Koordinaten (*X,Y*). Die Größe bleibt dabei unverändert. Wird der Zusatz *modus* verwendet, kann damit der Grafikmodus bestimmt werden. Bei Auslassen von *modus* wird im Replace-Modus gezeichnet.

| | |
|---|---|
| Modus 0 | alten Hintergrund löschen |
| Modus 1 | neues Bild AND Hintergrund |
| Modus 2 | neues Bild AND NOT Hintergrund |
| Modus 3 | neues Bild (ersetzt den Hintergrund, entspricht Graphmode 1) |
| Modus 4 | (NOT neues Bild) AND Hintergrund (entspricht Graphmode 4) |
| Modus 5 | Hintergrund (es geschieht nichts) |
| Modus 6 | neues Bild XOR Hintergrund (entspricht Graphmode 3) |

| | |
|---|---|
| Modus 7 | neues Bild OR Hintergrund (entspricht Graphmode 2) |
| Modus 8 | NOT (neues Bild OR Hintergrund) |
| Modus 9 | NOT (neues Bild XOR Hintergrund) (revers XOR) |
| Modus 10 | NOT Hintergrund (Hintergrund revers) |
| Modus 11 | neues Bild OR (NOT Hintergrund) |
| Modus 12 | NOT neues Bild (revers transparent) |
| Modus 13 | (NOT neues Bild) OR Hintergrund |
| Modus 14 | NOT (neues Bild AND Hintergrund) |
| Modus 15 | 1 (vollfarbig) |

| | |
|---|---|
| Keine Maustaste: | Neues Bild einmal zeichnen |
| Linke Maustaste: | Zweimal auf gleiche Position |
| Beide Maustasten: | Nächsten Grafikmodus einschalten |

# 7. Datenumwandlung

## ASC — Textzeichen -> ASCII-Wert

**ASC("Zeichen")**
(ermittelt den ASCII-Wert des angegebenen Zeichens)

**ASC("String")**
(ermittelt den ASCII-Wert des ersten Zeichens des angegebenen Strings)

**ASC(Stringvariable)**
(ermittelt den ASCII-Wert des ersten Zeichens des, in der angegebenen Variablen enthaltenen Strings)

Es kann der ASCII-Code eines Textzeichens ermittelt werden. *ASC* bildet die Umkehrfunktion zu *CHR$*. Soll der ASCII-Wert eines einzelnen Zeichens ermittelt werden, ist dieses Zeichen in Anführungszeichen innerhalb der auf *ASC* folgenden Klammer anzugeben. Auf die gleiche Art kann auch ein mehrstelliger String angegeben werden. In diesem Fall wird nur der ASCII-Wert des ersten Zeichens dieses Strings zurückgegeben. Ist der angegebene String leer ("""), wird von der Funktion der Wert 0 geliefert. *ASC* läßt sich wahlweise direkt ausgeben, einer numerischen Variablen übergeben oder in eine Bedingungsabfrage einbinden.

## BIN$ Numerisch -> Binär

**BIN$(Ausdruck)**

Wandelt im Dezimal-, Hexadezimal-, oder Oktal-System angegebene Werte in einen String mit der binären Darstellungsweise um. *Ausdruck* steht für eine Variable, einen Wert oder numerischen Ausdruck in einem der anderen drei Zahlensysteme. Der mit *BIN$* ermittelte String kann mit einem PRINT-Befehl direkt ausgegeben, einer Stringvariablen übergeben oder in einen Stringausdruck eingebunden werden. Will man Werte im Binär-Format angeben, so kann der Vorsatz *&X* (z.B. &X10011101) verwendet werden. Auf diese Art angegebene Binärzahlen werden vom Interpreter ebenso akzeptiert, wie Werte, die in einem der anderen Syteme dargestellt werden. Es sind auf diese Art allerdings nur Integerwerte darstellbar.

## CHR$ ASCII -> Textzeichen

**CHR$(Wert)**

Es wird ein String mit dem dem angegebenen ASCII-Wert entsprechenden Textzeichen gebildet. *CHR$* bildet die Umkehrfunktion zu *ASC*. Die ersten 32 ASCII-Werte (0-31) sind durch Steuerzeichen belegt und können durch *CHR$* nicht dargestellt werden. Es sind insgesamt (incl. der Steuerzeichen) 256 ASCII-Zeichen (0-255) vorhanden. Wird ein größerer ASCII-Wert als 255 übergeben, so wird das Zeichen ermittelt, das dem angegebenen *Wert MOD 256* entspricht. *CHR$* läßt sich wahlweise direkt ausgeben, einer Stringvariablen übergeben oder in einen Textausdruck einbinden.

**CVI(2-Zeichen-Stringausdruck)**
**CVL(4-Zeichen-Stringausdruck)**
**CVS(4-Zeichen-Stringausdruck)**
**CVF(6-Zeichen-Stringausdruck)**
**CVD(8-Zeichen-Stringausdruck)**

Es werden Zeichenketten in entsprechende Formatzahlen umgewandelt. Es wird die der jeweiligen Funktion entsprechende Zeichenanzahl von *Stringausdruck* gemäß der jeweiligen Formatvorgabe in eine Zahl umgewandelt.

**CVI** wandelt 2-Zeichen-Stringausdruck in einen 16-Bit-Integerwert

**CVL** wandelt 4-Zeichen-Stringausdruck in einen 32-Bit-Integerwert

**CVS** wandelt 4-Zeichen-Stringausdruck im Atari-BASIC-Format in eine Realzahl

**CVF** wandelt 6-Zeichen-Stringausdruck im GfA-BASIC-Format in eine Realzahl

**CVD** wandelt 8-Zeichen-Stringausdruck im MBASIC-Format in eine Realzahl

*Stringausdruck* kann als Text direkt angegeben oder als Stringvariable übergeben werden. Das gelieferte Ergebnis kann wahlweise direkt ausgegeben, einer Variablen übergeben oder in eine Bedingungsabfrage eingebunden werden. Diese Funktionen bilden die Umkehrfunktionen zu *MKI$ / MKL$ / MKS$ / MKF$ / MKD$*.

## HEX$ Numerisch -> Hexadezimal

**HEX$(Ausdruck)**

Wandelt im Dezimal-, Binär-, oder Oktalsystem angegebene Werte in einen String mit der hexadezimalen Darstellungsweise um. *Ausdruck* steht für eine Variable, einen Wert oder numerischen Ausdruck in einem der anderen drei Zahlensysteme. Der mit *HEX$* ermittelte String kann mit einem PRINT-Befehl direkt ausgegeben, einer Stringvariablen übergeben oder in einen Stringausdruck eingebunden werden. Will man Werte im Hexadezimal-Format angeben, so kann der Vorsatz *&H* (z.B. &HE4FA1B) verwendet werden. Auf diese Art angegebene Hexadezimalzahlen werden vom Interpreter ebenso akzeptiert, wie Werte, die in einem der anderen Syteme dargestellt werden. Es sind auf diese Art allerdings nur Integerwerte darstellbar.

## MKI$, MKL$, MKS$, MKF$, MKD$ Formate

**MKI$(16-Bit-Integer-Zahl)**
**MKL$(32-Bit-Integer-Zahl)**
**MKS$(Realzahl)**
**MKF$(Realzahl)**
**MKD$(Realzahl)**

Es wird eine beliebige Zahl in die, dem jeweiligen Format entsprechende Zeichenkette umgewandelt. Es wird die dem jeweiligen Format entsprechende Zahl in einen der Wertgröße und dem gewünschten Format entsprechenden Stringausdruck umgewandelt.

MKI$ wandelt 16-Bit-Integerwert in einen 2-Zeichen-Stringausdruck

MKL$ wandelt 32-Bit-Integerwert in einen 4-Zeichen-Stringausdruck

MKS$ wandelt eine Realzahl in einen 4-Zeichen-Stringausdruck im Atari-BASIC-Format

MKF$ wandelt eine Realzahl in einen 6-Zeichen-Stringausdruck im GfA-BASIC-Format

MKD$ wandelt eine Realzahl in einen 8-Zeichen-Stringausdruck im MBASIC-Format

Der jeweilige Wert kann als numerischer Ausdruck oder als Variable übergeben werden. Der gelieferte String kann wahlweise direkt ausgegeben, einer Stringvariablen übergeben oder in einen Stringausdruck eingebunden werden. Diese Funktionen bilden die Umkehrfunktionen zu *CVI$ / CVL$ / CVS$ / CVF$ / CVD$.*

## OCT$ Numerisch -> Oktal

**OCT$(Ausdruck)**

Wandelt im Dezimal-, Hexadezimal- oder Binärsystem angegebene Werte in einen String mit der oktalen Darstellungsweise um. *Ausdruck* steht für eine Variable, einen Wert oder numerischen Ausdruck in einem der anderen drei Zahlensysteme. Der mit *OCT$* ermittelte String kann mit einem PRINT-Befehl direkt ausgegeben, einer Stringvariablen übergeben oder in einen Stringausdruck eingebunden werden. Will man Werte im Oktal-Format angeben, so kann der Vorsatz *&O* (z.B. &O25516277) verwendet werden. Auf diese Art angegebene Oktalzahlen werden vom Interpreter ebenso akzeptiert, wie Werte, die in einem der anderen Syteme dargestellt werden. Es sind auf diese Art allerdings nur Integerwerte darstellbar.

## STR$ Numerisch -> String

**STR$(Wert)**

Es lassen sich numerische Werte in einen Textausdruck umwandeln. *Wert* kann als beliebiger Wert in jedem beliebigen Zahlensystem angegeben werden. Als Hexadezimal-, Binär- oder Oktalzahl angegebene Werte werden vorher in das Dezimalformat umgewandelt. Der entstehende String hat anschließend die Länge, die der Anzahl der Ziffern des übergebenen Wertes im Dezimalformat entspricht. Die gelieferte Ziffernfolge ist anschließend keine Zahl mehr, die einen Wert darstellt, sondern lediglich ein Textstring, der die einzelnen Ziffern des Wertes als Textzeichen enthält. Er kann direkt ausgegeben, an eine Stringvariable übergeben oder in einen Textausdruck eingebunden werden. *STR$* bildet die Umkehrfunktion zu *VAL*.

## VAL String -> Numerisch

**VAL(Stringausdruck)**

Wandelt alle am Anfang eines Strings stehenden Zeichen, die sich zur Darstellung numerischer Werte eignen, in eine dezimale Realzahl um. *Stringausdruck* ist eine beliebige Zeichenkette oder Stringvariable, deren Inhalt vom Anfang ausgehend daraufhin untersucht wird, ob Textzeichen enthalten sind, die einen Wert in einem der vier Zahlensysteme darstellen. Die Suche wird abgebrochen, wenn das Stringende erreicht ist oder die Funktion auf ein Textzeichen trifft, welches nicht wandelbar ist. Der evtl. gelesene Wert wird in eine dezimale Realzahl umgewandelt. Ist das erste Zeichen des Strings ein nicht wandelbares Textzeichen oder ist der String leer, wird eine Null zurückgegeben. Der ermittelte Wert kann wahlweise direkt ausgegeben, einer Variablen übergeben oder in eine Bedingungsabfrage eingebunden werden.

**VAL?(Stringausdruck)**

Ermittelt, vom Anfang des Strings aus gezählt, die Anzahl der Zeichen eines Strings, die in numerische Werte konvertiert werden können. *Stringausdruck* steht für einen beliebigen Textausdruck oder eine Stringvariable, die auf die Anzahl ihrer, in numerische Werte andelbarer Zeichen untersucht werden soll. Es wird immer nur so weit untersucht, wie Zeichen auftreten, die wandelbar sind. Der gelieferte Wert kann wahlweise direkt ausgegeben, an eine Variable übergeben oder in eine Bedingungsabfrage eingebunden werden.

# 8. Feld-, Speicher und Zeigerbefehle

## 8.1 Feldoperationen

### ARRAYFILL [AR] Feld mit Wert belegen

**ARRAYFILL Array(),x**

Belegt alle Elemente eines Feldes mit einer Konstanten. *Array* ist der Name einer beliebigen numerischen oder boolschen Feldvariablen, die dimensioniert sein muß. Alle Elemente des Feldes werden mit der Konstanten *X* belegt. *X* kann eine Zahl oder eine numerische Variable sein. Das Füllen von Textfeldern ist nicht möglich. Der angegebene Wertetyp *X* muß mit dem Feldtyp übereinstimmen.

### DIM [DI] Feld dimensionieren

**DIM V_name(index1,index2,index3...)**
**DIM V_name1(indizes),V_name2(indizes),...**

Legt die Dimension eines oder mehrerer Felder fest und reserviert hierfür Speicherplatz. *indizes* ist eine Liste von Integerwerten oder numerischer Variablen. Der Befehl initialisiert ein beliebiges Feld unter dem Namen *V_name*, *V_name1*..., indem er anhand der Anzahl der *indizes* festlegt, wie viele Dimensionen das Feld erhalten soll (z.B. zweidimensional = Matrix). Der Index selber besagt, wieviele Elemente pro Dimension zugelassen sind. Es wird also eine Indexobergrenze (kleinster Indexwert = 0) gesetzt und der nötige Speicherplatz reserviert. Bei einem mehrdimensionalen Feld ist die Anzahl der Elemente auf 65535, bei eindimensionalen Feldern nur durch die Größe des Hauptspeicherplatzes begrenzt. Die Dimensionierung eines Feldes kann nur mit *ERASE* wieder gelöscht werden.

## DIM? Menge der Feldelemente ermitteln

**DIM?(Array())**

Übergibt die Anzahl aller Elemente eines beliebigen Feldes. *Array* ist der Name der Feldvariablen, deren Elementeanzahl ermittelt, bzw. übergeben wird. Wurde das Feld vorher nicht dimensioniert, wird der Wert 0 geliefert. Achtung: Die Indizierung jeder Dimension beginnt mit Null.

## ERASE [ER] Feld löschen

**ERASE Array()**

Löscht eine Feldvariable und hebt ihre Dimensionierung auf. *Array* steht hier für den Namen des Feldes, das gelöscht werden soll. Der dafür reservierte Speicher wird wieder freigegeben und kann neu belegt werden.

## 8.2 Speicheroperationen

### D/L/PEEK — Speicherinhalt auslesen

**PEEK(Adresse)**
(liest ein Byte)

**DPEEK(gerade Adresse)**
(liest zwei Byte (Word))

**LPEEK(gerade Adresse)**
(liest vier Byte (Long))

Liest im jeweiligen Format die angegebene Speicheradresse. Es ist darauf zu achten, daß bei *DPEEK* und *LPEEK* jeweils eine gerade Adresse angegeben wird, da sonst eine Fehlermeldung erscheint. *Adresse* kann als Wert, numerischer Ausdruck oder Variable übergeben werden. Der jeweils gelieferte Wert kann wahlweise direkt ausgegeben, einer Variablen übergeben oder in eine Bedingungsabfrage eingebunden werden. Diese Funktionen bilden die Umkehrfunktionen zu *POKE / DPOKE / LPOKE*, bzw. *SPOKE / SDPOKE / SLPOKE*.

**POKE Adresse,Byte**
(schreibt ein Byte)

**DPOKE gerade Adresse,Word**
(schreibt zwei Byte)

**LPOKE gerade Adresse,Long**
(schreibt vier Byte)

Übergibt im User-Modus den angegebenen Wert im jeweiligen Format an die angegebene Speicherstelle. *POKE* ersetzt den Inhalt eines einzelnen Bytes durch den Integerwert *Byte* im Bereich von -256 bis 255.

*DPOKE* ersetzt den Inhalt zweier aufeinander folgender Bytes durch den Integerwert *Word* im Bereich von -65536 bis 65535. *LPOKE* ersetzt den Inhalt von vier aufeinander folgenden Bytes durch den Integerwert "Long" im Bereich von -2147483648 bis 2147483647. Es ist darauf zu achten, daß an *DPOKE* und *LPOKE* nur gerade Adressen übergeben werden, da sonst eine Fehlermeldung erscheint. Beide Parameter können als Wert, numerischer Ausdruck oder als Variable übergeben werden. Diese Befehle bilden die Umkehrfunktionen zu *PEEK / DPEEK / LPEEK* im User-Modus.

## S/SD/SLPOKE [SP,SD,SL] Supervisorpoke

**SPOKE Adresse,Byte**
(schreibt ein Byte)

**SDPOKE gerade Adresse,Word**
(schreibt zwei Byte)

**SLPOKE gerade Adresse,Long**
(schreibt vier Byte)

Übergibt im Supervisor-Modus den angegebenen Wert im jeweiligen Format an eine privilegierte Speicherstelle. Für diese Befehle gelten dieselben Ausführungen wie zu *POKE / DPOKE / LPOKE*. Der Unterschied ist, daß hiermit auf sogenannte "privilegierte" Speicherbereiche (Supervisor-Modus) zugegriffen und deren Inhalte verändert werden kann. Diese Befehle bilden die Umkehrfunktionen zu *PEEK / DPEEK / LPEEK* im Supervisor-Modus.

## C: Maschinenprogramm (C-compiliert) aufrufen

**C:V_name(Parameterliste)**

Es kann ein in C kompiliertes Maschinenprogramm aufgerufen werden. Es ist die Adresse der Maschinenroutine und, gegebenenfalls, eine *Parameterliste* in Klammern zu übergeben. Sind keine Parameter vorhanden, ist eine leere Klammer () zu verwenden. Sollen 32-Bit-Parameter übergeben werden, ist dem jeweiligen Parameter das Kürzel *L:* (=Long) voranzustellen. Ohne Angabe einer Kennung werden Words übergeben. Nach Rückkehr zum BASIC wird der Inhalt von D0 als Longword geliefert. Dieser Wert kann erfahren, bzw. ausgewertet werden, indem der C:-Aufruf einer Variablen übergeben, direkt ausgegeben oder in Bedingungsabfragen eingebunden wird.

```
        A%=C:V_name()
bzw.    Print C:V_name()
bzw.    If C:V_name()=Wert
```

## CALL [CA] Maschinenprogramm aufrufen

**CALL V_name**
**CALL V_name(Parameterliste)**

Es kann ein assembliertes Maschinenprogramm aufgerufen werden. Es ist in *V_name* die Adresse der aufzurufenden Maschinenroutine zu übergeben, sowie gegebenenfalls eine Liste, die die an die Routine zu übergebenden Parameter enthält.

## 8.3 Zeigeroperationen

Die Variablenorganisation im GfA-BASIC: Variablennamen ohne Kennung werden als Fließkommavariablen (reell) interpretiert. Dieser Variablentyp benötigt zu seiner Speicherung 6 Byte Speicherplatz. Im Dezimalformat kann eine Genauigkeit von bis zu maximal 11 Stellen eingehalten werden. Nimmt der ganzzahlige Anteil mehr als 11 Stellen ein, wird der Variablenwert in das Exponentialformat konvertiert. In diesem Format können dann Werte mit einem ganzzahligen Anteil von bis zu 154 Stellen erfaßt werden (größter darstellbarer Wert = 1.0E+154). Variablennamen mit der Kennung "%" (z.B. V_name%) gelten als Integervariablen. Jeder diesem Variablentyp zugeordnete Wert wird auf seinen ganzzahligen Anteil reduziert. D.h., evtl. auftretenden Nachkommastellen werden "integriert". Zu seiner Speicherung benötigt dieser Typ 4 Byte Speicherplatz. Es können Werte im Bereich von -2147483648 (= -2^31) bis +2147483647 (= 2^31-1) verarbeitet werden.

Variablennamen mit der Kennung *!* (z.B. *V_name!*) sind Boolsche Variablen. Dieser Variablentyp kann ausschließlich die Werte 0 (= false) und -1 (<> 0 = true) annehmen. Es werden zu seiner Speicherung 2 Byte benötigt.

Variablennamen mit der Kennung *$* (z.B. *V_name$*) werden als Textvariablen (Stringvariablen) interpretiert. Für diesen Variablentyp wird jeweils ein Descriptor von 6 Byte eingerichtet. In diesem Descriptor befindet sich in den ersten vier Byte als Longword die Adresse des ersten Zeichens des Strings und in den nächsten zwei Byte die Länge des Strings. Der String-Descriptor ist über den Befehl *ARRPTR(V_name$)*" zu ermitteln. Jedem String, der in einer Variablen gespeichert wird, wird ein 4 Byte langer Backtrailer (Anhänger) zugeordnet. Dieser beinhaltet die Anfangsadresse (Longword) des jeweils zugehörigen Descriptors. Hat der String eine ungerade Länge, wird zwischen Stringende und Backtrailer ein Füllbyte gesetzt, damit das Backtrailer-Longword auf jeden Fall eine gerade Adresse erhält.

Beispiel 1:

```
A$="BASIC"
Backtr.adr%=Varptr(A$)+Len(A$)+1
Descr.adr.v%=Lpeek(Backtr.adr%)
Descr.adr.a%=Arrptr(A$)
S.tringlen.l%=Len(A$)
S.tringlen.a%=Dpeek(Arrptr(A$))
S.tringadr.v%=Varptr(A$)
S.tringadr.a%=Lpeek(Arrptr(A$))
S.tring$=Mkl$(Lpeek(S.tringadr.v%))+Chr$(Peek(S.tringadr.v%+4))
Print "Stringlänge   : ";S.tringlen.a%''S.tringlen.l%
Print "Descr.-Adresse: ";Descr.adr.v%''Descr.adr.a%
Print "String-Adresse: ";S.tringadr.v%''S.tringadr.a%
Print "String: Füllbyte: Backtrailer:"
Print S.tring$'''Chr$(Peek(Backtr.adr%-1)),''Backtr.adr%
```

Für jedes Zeichen (ASCII 0-255 = 8 Bit) des Strings wird ein Byte zur Speicherung verwendet. Ein String mit einer Länge von beispielsweise 37 Zeichen nimmt ab seiner Anfangsadresse einen Speicherplatz von 37 + 1 Füllbyte + 4 Backtrailerbytes = 42 Bytes + zusätzliche 6 Descriptorbytes = 48 Bytes in Anspruch.

Mit jedem der angeführten Variablentypen lassen sich auch mehrdimensionale Felder (Arrays) bilden. Zur Bestimmung der Adressen der einzelnen Feldelemente und ihrer Größen (bei Stringfeldern), sowie zur Ermittlung der Dimensionsanzahl und deren einzelnen Elementmengen kann ebenfalls der Befehl *ARRPTR* verwendet werden. Dieser hat allerdings, je nachdem, ob es sich um ein numerisches oder alphanumerisches Feld handelt, unterschiedliche Funktionsweisen.

Der erste Feld-Descriptor gibt in seinen ersten vier Byte Auskunft über die Feldadresse. In den nächsten zwei Byte ist die Anzahl der Dimensionen enthalten.

Beispiel 2:

```
Dim Feld$(2,3,1)
Feld$(0,0,0)="AAAAAAAA"
Feld$(1,0,0)="BBBBBBBB"
Feld$(2,0,0)="CCCCCCCC"
Print "Feldadresse  : ";Lpeek(Arrptr(Feld$()))
Print "Anz.d.Dimens.: ";Dpeek(Arrptr(Feld$())+4)
```

Der erste PRINT-Befehl liefert eine Adresse, die von Fall zu Fall variiert. Der zweite PRINT-Befehl gibt die Zahl 3 aus, da 3 Dimensionen eingerichtet wurden (2,3,1).

Unter Feldadresse ist hier nicht die Adresse des ersten Byte des ersten Elementes der ersten Dimension zu verstehen, sondern ein weiterer Descriptor, der Informationen über die einzelnen Dimensionen liefert. Die Größe dieses Descriptors ist abhängig von den Dimensionen, die mit *DIM* vorgesehen wurden. Am Anfang dieses zweiten Descriptors stehen in jeweils 4 Byte die Dimensionstiefen, rückwärts. Wenn also 3 Dimensionen angegeben wurden, ist der erste Sektor dieses zweiten Descriptors 12 Byte lang. Wurden 4 Dimensionen eingerichtet, ist er 16 Byte lang usw.

Nach der Formel:

```
  Adresse des zweiten Descriptors
+ (Anzahl der Dimensionen * 4)
- (Dimensionsindex * 4)
```

können die einzelnen Elementemengen der mit *DIM* bestimmten Dimensionen ermittelt werden.

Fortsetzung zu Beispiel 2:

```
D_imdescript.%=Lpeek(Arrptr(Feld$()))
D_imanzahl%=Dpeek(Arrptr(Feld$())+4)
For I%=1 To D_imanzahl%
  D_imdeep%=D_imdescript.%+(4*D_imanzahl%)-(4*I%)
  Print "Tiefe der ";I%;". Dimension : ";Lpeek(D_imdeep%)
Next I%
```

```
Ausgabe: Tiefe der 1. Dimension: 3
         Tiefe der 2. Dimension: 4
         Tiefe der 3. Dimension: 2
```

Die Anzahl der Dimensionsunterteilungen ist immer um 1 größer, als mit *DIM* angegeben, da die Indizierung immer mit dem Element 0 der jeweiligen Dimension beginnt. Ab hier unterscheiden sich die Descriptoren für Stringfelder von denen für numerische Felder. Für Stringfelder schließt sich an diesen ersten Sektor, der in seiner Länge von der Anzahl der Dimensionen abhängig ist, zusätzlich eine Liste von Descriptoren an, die denselben Aufbau haben, wie der *ARRPTR* für Einzelstrings. In diesem Falle existieren für jedes Element der ersten Dimension eines Stringarrays ein separater Descriptor. Bei der im Beispiel verwendeten Dimensionierung sind es drei Descriptoren.

Deren Lage läßt sich nach der Formel:

```
  Adresse des zweiten Descriptors
+ (Anzahl der Dimensionen * 4)
+ (Elementindex der ersten Dimension * 6)
```

ermitteln.

Fortsetzung zu Beispiel 2:

```
D_imdeep.0%=Lpeek(D_imdescript.%+(4*D_imanzahl%)-4)
S_tringptr%=D_imdescript.%+(4*D_imanzahl%)
For I%=0 To D_imdeep.0%-1
  Print "String-Adresse ";I%;"/0/0 : ";
  Print Lpeek(S_tringptr%+(I%*6))
  Print "String-Länge   ";I%;"/0/0 : ";
  Print Dpeek(S_tringptr%+4+(I%*6))
  Print "Descr.Adresse in Backtrailer ";I%;"/0/0 : ";
  Print Lpeek(Lpeek(S_tringptr%+(I%*6))+Len(Feld$(I%,0,0)))
  Print "String-Adresse über VARPTR :";
  Print Varptr(Feld$(I%,0,0))
Next I%
```

Die Ausgabe fällt hier, je nach konkreter Situation, unterschiedlich aus.

Diese Feldstringpointer weisen in ihren ersten vier Bytes auf Adressen, die ca. 50 Kilobyte unterhalb des Bildschirmspeichers liegen. In GfA-BASIC werden alle Arrays rückwärts angelegt. D.h., daß das erste Byte des jeweiligen Strings im Speicher an der höchsten Adresse liegt und alle anderen Zeichen des Strings sich der Reihe nach rückwärts daran anschließen. Die Adressen aller Variablen (auch von Feldvariablen-Elementen) lassen sich zusätzlich auch durch den Befehl *VARPTR* ermitteln.

## ARRPTR — Felddescriptor-Adresse ermitteln

**ARRPTR(Stringname$)**
**ARRPTR(Feldname())**

Liefert die Anfangsadresse von String- bzw. Feld-Descriptor. Siehe Erläuterungen am Kapitelanfang unter *Variablenorganisation*. Siehe auch Beispielprogramm 2 am Kapitelanfang.

## VARPTR — Variablenadresse ermitteln

**VARPTR(V_name)**

Es kann bei numerischen Variablen deren Adresse bzw. bei Stringvariablen die Adresse des ersten Zeichens ermittelt werden. *V_name* steht für jede beliebige Variable, deren Speicheradresse geliefert werden soll. (Siehe auch Beispielprogramm 2 am Kapitelanfang.)

# 9. Dialogabwicklung

## ALERT [A] . Alertbox erstellen

**ALERT Icon%,B_oxtext$,Def_Button%,Buttontext$, Backvar%**

Aufruf eines Formulars zur beschränkten Abwicklung von Anwenderdialogen. Der Befehl *ALERT* besteht aus fünf Komponenten:

**Icon%**
0 = kein Symbol
1 = Ausrufungszeichen
2 = Fragezeichen
3 = STOP-Schild

**B_oxtext$** Hier wird der eigentliche Text (Mitteilung/ Frage) an die Funktion übergeben. Das Zeichen | (Pipe) gilt darin als Trennungszeichen zwischen den einzelnen Zeilen. Es können insgesamt 4 Zeilen zu maximal je 30 Zeichen dargestellt werden. Der Text kann direkt in den Befehl geschrieben oder auch als Stringvariable übergeben werden.

**Def_Button%**
Es wird die Nummer (1,2,3) des Buttons übergeben, welcher außer durch Mausklick auch durch die <RETURN>-Taste (default) bestätigt werden kann. Dieser Button wird in der Box stark umrandet gezeichnet (0=kein Default-Button).

**Buttontext$**
Hierdurch erfolgt die Beschriftung der Buttons. Jeder Button kann mit jeweils maximal 8 Zeichen beschriftet werden. Auch hier gilt das |-Zeichen als Trennstrich zwischen den einzelnen Buttons.

**Backvar%** An letzter Stelle wird eine frei benennbare numerische Variable angegeben, in welcher die Nummer des bestätigten Button (1,2,3) von der Funktion zurückgegeben wird. Die Texte können auch direkt, sowie die Werte auch als Variablen in den Befehl eingesetzt werden.

## FILESELECT [FILESE] Datei auswählen

**FILESELECT "Pfadangabe","Vorauswahlangabe",Backvar$**

Erstellt ein Dialogformular zur Dateiauswahl und übergibt den gewählten Pfad. Es werden drei Parameter übergeben. Der erste ist ein String oder eine Stringvariable, in welcher der Pfad zu der gesuchten Datei angegeben wird. Im zweiten kann gegebenenfalls ein Dateiname angegeben werden, der bei Aufruf der Box in die Eintragszeile rechts unter *Auswahl* eingesetzt wird. Hier ist zu beachten, daß tatsächlich nur maximal 12 Zeichen (incl. Punkt) verwendet werden (8 für Dateinamen / 1 für Trennpunkt / 3 für Extension). Es kann auch "" eingesetzt werden oder nur die Extension der Datei. Der dritte Parameter ist eine beliebig zu benennende Stringvariable, die den kompletten Namen der vom Anwender ausgewählten Datei (mit Pfadbezeichnung) aufnehmen soll. Bei Rückkehr zum Programm sind fünf verschiedene Eintragsmöglichkeiten in diese Variable gegeben. Wurde tatsächlich vom Anwender eine Datei entweder durch Doppelklick auf den entsprechenden Namen im Directory-Fenster oder einfachen Klick darauf und anschließendem Klick in die Ok-Box gewählt, steht ihr Name auch anschließend in der Rückgabevariablen. Wurde jedoch ohne Auswahl auf die OK-Box geklickt, gibt es drei Varianten. Entweder es wurde vorher ein Dateiname übergeben, der in *Auswahl* steht und die Ok-Box wurde bedient, dann steht dieser Name auch in der Rückgabevariablen. Oder es wurde nur die Extension übergeben, dann steht die gewählte Extension in der Variablen. Wurde keine Auswahl getroffen und auch kein Name oder Extension übergeben oder diese vom Anwender in der Box gelöscht und die Ok-Box angeklickt, wird in der Rückgabevariablen ein BACKSLASH (\) eingetragen. Wird dagegen *Abbruch* angeklickt, ist die Variable absolut leer ("").

## MOUSE [M] Mausstatus ermitteln (gesamt)

**MOUSE Xpos,Ypos,Tasten**

Fragt die Position der Maus und den Zustand der beiden Maustasten ab. In den Variablen *Xpos* und *Ypos* wird die X-, bzw. Y-Koordinate des Mauszeigers, sowie in der Variablen *Tasten* der Status der Maustasten übergeben.

| | |
|---|---|
| 0 = | keine Taste gedrückt |
| 1 = | linke Taste gedrückt |
| 2 = | rechte Taste gedrückt |
| 3 = | beide Tasten gedrückt |

## MOUSEX/Y/K Mausstatus

**MOUSEX**
(X-Position)

**MOUSEY**
(Y-Position)

**MOUSEK**
(Maustastenstatus)

Fragt die Position der Maus und den Zustand der beiden Maustasten jeweils einzeln ab. Die Befehle bewirken eine Einzelabfrage des jeweils gewünschten Mausstatus. Die gelieferten Werte können wahlweise in eine Variable übergeben, in Bedingungsabfragen eingebunden oder direkt ausgegeben werden.

# 10. Window-Programmierung

## CLEARW [CLE W] Fensterinhalt löschen

**CLEARW Handle**

Löscht den Fensterinhalt eines GEM-Windows. Die Fensterkennung *Handle* bestimmt die Nummer des Windows (0...4), dessen Inhalt gelöscht werden soll. Handelt es sich um das aktuelle Fenster, kann hierzu genauso der Befehl *CLS* verwendet werden.

## CLOSEW [CL W] Fenster schließen

**CLOSEW Handle**

Schließt ein GEM-Window. Die Fensterkennung *Handle* bestimmt die Nummer des Windows (0...4), welches geschlossen wird.

## FULLW [FU] Fenster auf Bildschirmgröße

**FULLW Handle**

Vergrößert ein GEM-Window auf die volle Größe des Bildschirms bis an die Menüzeilen-Aussparung. Die Fensterkennung *Handle* bestimmt die Nummer des Windows (1...4), welches auf die X-/Y-Koordinaten 0/20 , 639/20 , 0/399 , 639/399 (=Window-Randpunkte vergrößert wird. Der restliche Bereich bleibt einer Menü-Zeile vorbehalten (mehr hierzu siehe unter *MENU*).

## INFOW [INF] Fenster-Informationszeile

**INFOW Handle,Stringausdruck**

Bestimmt, ob eine Window-Infozeile geöffnet werden soll und legt ihren Inhalt fest. Ein GEM-Window kann eine Informationszeile enthalten, die sich unterhalb der Window-Titelzeile befindet. Die Fenster-Kennung *HANDLE* bestimmt die Nummer des anzusprechenden Windows. Der Infotext *Stringausdruck* kann entweder als beliebiger Text in Anführungszeichen oder als Stringvariable übergeben werden. Soll diese Infozeile benutzt werden, muß sie mit dem Befehl *INFOW* angelegt werden, bevor das entsprechende Fenster mit dem Befehl *OPENW* geöffnet wird. Wurde ein Fenster ohne Infozeile geöffnet, so wirkt sich der Befehl *INFOW* erst auf das Fenster aus, wenn dieses mit *CLOSEW* geschlossen und mit *OPENW* erneut geöffnet wird. Genauso muß man vorgehen, wenn man die Infozeile löschen will. Hierzu muß *Stringausdruck* ein Leerstring ("") sein.

## OPENW [O W] Fenster öffnen

**OPENW Handle,X_pos,Y_pos**
(Schnittpunkt bestimmen)

**OPENW Handle**
(öffnen/aktualisieren)

Öffnet ein GEM-Window bzw. bewirkt, daß dieses als aktuelles Fenster dargestellt wird. Mit der Kennziffer *Handle* wird bestimmt, welches Fenster angesprochen wird. Die Zahlen 1,2,3 und 4 stehen hierbei für jedes der 4 möglichen Windows. Die Fenster werden hier ohne Überlappungen nebeneinander angeordnet:

| | |
|---|---|
| Window 1 | Window 2 |
| Window 3 | Window 4 |

Dabei berühren sich die Fenster, am Bildschirmpunkt, der mit dem Koordinatenpaar *X_pos/Y_pos* bestimmt wird. Dieser Schnittpunkt braucht nur einmal bestimmt werden und kann beim Öffnen oder Aktualisieren eines beliebigen Windows durch Angabe der neuen Werte *X_pos* und *Y_pos* verändert werden. Hiermit verschieben sich die Berührungsachsen der Fenster auf den neuen Schnittpunkt. Wurde ein Fenster bereits geöffnet und ist es nicht aktuell, so kann es mit *OPENW* zum aktuellen Fenster gemacht werden.

Beim Öffnen eines beliebigen Fensters, wird automatisch ein nicht sichtbares Fenster mit dem *Handle* = 0 geöffnet. Bei Programmende oder im Falle, daß der gesamte TOS-Ausgabebildschirm aktiviert werden soll, ist dieses Window vorher mit *CLOSEW* 0 zu schließen. Ist kein weiteres GEM-Window geöffnet, kann mit dem Befehl *Openw 0,X_pos,Y_pos* der Nullpunkt für Grafikausgaben verändert werden.

```
Openw 0
Deffill ,2,4
Pbox 0,0,639,380
For I=1 To 4
  Dpoke Windtab+2+(I-1)*12,&X111111111111
  Titlew I," TITEL-Zeile "+Str$(I)
  Infow I," INFO-Zeile "+Str$(I)
  Openw I
  Clearw I
Next I
Count=4
On Menu Message Gosub M.essage
Do
  A$=Inkey$
  If A$=>"1" And A$=<"4"
    Openw Val(A$)
    Clearw Val(A$)
    Inc Count
  Endif
  On Menu
  Mouse X,Y,K
  Xr=Dpeek(Windtab+8+(Menu(4)-1)*12)-20
  Yr=Dpeek(Windtab+10+(Menu(4)-1)*12)-60
  If K=1 And X=>0 And Y=>0 And X<Xr And Y<Yr
```

```
    Print X''Y
  Endif
Loop
Procedure M.essage
  If Menu(1)=20
    Print "Bereich ";Menu(4);": ";Menu(5)'Menu(6)'Menu(7)'Menu(8)
  Endif
  If Menu(1)=21
    Infow Menu(4),"(TOPPER) Window "+Str$(Menu(4))+" gewählt"
    Openw Menu(4)
    Clearw Menu(4)
  Endif
  If Menu(1)=22
    Closew Menu(4)
    Dec Count
    If Count=0
      Alert 2,"Programm-Ende ?",1,"OKAY|NEIN",D%
      If D%=1
        Closew 0
        Cls
        Edit
      Else
        Alert 1,"Fenster öffnen:|Tasten 1 - 4",1,"OKAY",D%
      Endif
    Endif
  Endif
  If Menu(1)=23
    Closew Menu(4)
    Dpoke Windtab+4+(Menu(4)-1)*12,0
    Dpoke Windtab+6+(Menu(4)-1)*12,19
    Dpoke Windtab+8+(Menu(4)-1)*12,639
    Dpoke Windtab+12+(Menu(4)-1)*12,380
    Infow Menu(4),"(FULLER) Volle Größe !"
    Openw Menu(4)
    Clearw Menu(4)
  Endif
  If Menu(1)=24
    Infow Menu(4),"(ARROWS) Pfeil:"+Str$(Menu(5))
  Endif
  If Menu(1)=25
    Infow Menu(4),"(SLIDE)H-Schieber : "+Str$(Menu(5))
  Endif
  If Menu(1)=26
    Infow Menu(4),"(SLIDE)V-Schieber : "+Str$(Menu(5))
```

```
  Endif
  If Menu(1)=27
    Dpoke Windtab+8+(Menu(4)-1)*12,Menu(7)
    Dpoke Windtab+10+(Menu(4)-1)*12,Menu(8)
    Closew Menu(4)
    Infow Menu(4),"(SIZER) Fenstergröße verändert"
    Openw Menu(4)
    Clearw Menu(4)
  Endif
  If Menu(1)=28
    Dpoke Windtab+4+(Menu(4)-1)*12,Menu(5)
    Dpoke Windtab+6+(Menu(4)-1)*12,Menu(6)
    Dpoke Windtab+8+(Menu(4)-1)*12,Menu(7)
    Dpoke Windtab+10+(Menu(4)-1)*12,Menu(8)
    Closew Menu(4)
    Infow Menu(4),"(MOVER) Fenster wurde bewegt"
    Openw Menu(4)
    Clearw Menu(4)
  Endif
Return
```

## TITLEW [TIT] Fenster-Titelzeile bestimmen

**TITLEW Handle,Stringausdruck**

Bestimmt oder verändert die Überschrift eines GEM-Windows. Einem Fenster kann eine Überschrift gegeben werden. Diese befindet sich bei einem geöffnetem Window innerhalb des grauen MOVE-Balkens. Der Befehl *TITLEW* funktioniert genauso, wie unter *INFOW* beschrieben.

Ausnahme:

```
Stringausdruck = "  " (Leerstring) -> Keine Überschrift
Stringausdruck = ""   (Nullstring) -> Fenster läßt sich nicht mehr
                                      bewegen.
```

**DPEEK (WINDTAB)**
**DPOKE WINDTAB,Wert**

Startadresse eines Speicherbereichs, aus welchem Daten zur Verwaltung von GEM-Windows gelesen werden können, bzw. dessen Inhalt nach Bedarf und Notwendigkeit verändert werden kann. Die Window-Tabelle *Windtab* ist eine reservierte Variable. In ihr werden 16-Bit-Werte abgelegt, die über die jeweiligen Zustände der Windows Auskunft geben.

Die WINDTAB-Tabelle hat folgenden Aufbau:

```
Windtab      =  Window-Handle 1
Windtab+2    =  Attribute für Window 1 (s.o.)
Windtab+4    =  X-Koordinate für Window 1
Windtab+6    =  Y-Koordinate für Window 1
Windtab+8    =  Breite Window 1
Windtab+10   =  Höhe Window 1
Windtab+12
  bis        =  entsprechende Angaben für Window 2
Windtab+22
Windtab+24
  bis        =  entsprechende Angaben für Window 3
Windtab+34
Windtab+36
  bis        =  entsprechende Angaben für Window 4
Windtab+46
Windtab+52   =  Bildschirm-X-Koordinate
Windtab+54   =  Bildschirm-Y-Koordinate
Windtab+56   =  Bildschirmbreite
Windtab+58   =  Bildschirmhöhe
Windtab+60   =  X-Schnittpunkt der vier Fenster
Windtab+62   =  Y-Schnittpunkt der vier Fenster
Windtab+64   =  X-Nullpunkt für Grafikbefehle
Windtab+68   =  Y-Nullpunkt für Grafikbefehle
```

Es kann außerdem in vier Worten (je 16 Bit) dieser Tabelle bestimmt werden, welche Randobjekte des jeweiligen Fensters aktiviert werden sollen. Dies geschieht durch:

```
Dpoke Windtab+2, 12-Bit-Wert      für Window 1
Dpoke Windtab+14,12-Bit-Wert      für Window 2
Dpoke Windtab+26,12-Bit-Wert      für Window 3
Dpoke Windtab+38,12-Bit-Wert      für Window 4
```

Diesen WINDTAB-Stellen wird ein 12-Bit-Wert übergeben. Durch einzelnes Setzen oder Addition der Bitwerte können die Komponenten bestimmt werden:

```
1    = der NAME
2    = das CLOSE-Feld links oben
4    = das FULL-Feld rechts oben
8    = der MOVE-Balken oberer Randbereich
16   = die Infozeile unter dem MOVE-Balken
32   = das SIZE-Feld rechts unten
64   = der Aufwärts-Pfeil rechts
128  = der Abwärtspfeil rechts
256  = der VSLIDE-Balken rechts (Vertikalschieber)
512  = der Links-Pfeil unten
1024 = der Rechts-Pfeil unten
2048 = der HSLIDE-Balken unten (Horizontalschieber)
```

## MENU(Index) Event-Puffer

**MENU(Index)**

Es können, je nach aufgetretenem Ereignis, die für die jeweilige Aktion relevanten Daten ermittelt werden. Hinter diesem Befehl verbirgt sich ein eindimensionales numerisches Feld, in welchem vom Interpreter permanent verschiedene Daten zu dem jeweils eingetretenen Ereignis abgelegt werden. Jedes Element dieses Feldes kann unter Angabe von *Index* ausgelesen werden. In den Punkten 1-8 können die Bedeutungen der Einträge von Fall zu Fall variieren.

Die Inhalte von *MENU(Index)*:

```
Menu(-1) = Adresse des aktuellen Menü-Objektbaumes
Menu(0)  = Index des gewählten Menüpunktes
Menu(1)  = Kenn-Nummer des jeweiligen Ereignisses
      10 = (WM_SELECTED) Pulldown-Menü wurde angewählt.
           Menu(4) = Titelindex
           Menu(5) = Menüpunktindex
      20 = (WM_REDRAW) Windowbereich neu zeichnen.
           Menu(4) = Handle
           Menu(5) = X-Koordinate
           Menu(6) = Y-Koordinate
           Menu(7) = Breite
           Menu(8) = Höhe
      21 = (WM_TOPPED) ein Window soll aktiviert werden.
           Menu(4) = Handle
      22 = (WM_CLOSED) Schließfeld links oben wurde gewählt.
           Menu(4) = Handle
      23 = (WM_FULLED) Vollfeld rechts oben wurde gewählt.
           Menu(4) = Handle
```

24 = (WM_ARROWED) Window-Inhalt soll gescrollt werden.
Es wurde einer der vier Pfeile oder einer der beiden grauen Scrollbalken gewählt.
Menu(4) = Handle
Menu(5) = Index des gewählten Objektes:
0 = ganze Seite nach oben
1 = ganze Seite nach unten
2 = eine Zeile nach oben
3 = eine Zeile nach unten
4 = ganze Seite nach links
5 = ganze Seite nach rechts
6 = ein Zeichen nach links
7 = ein Zeichen nach rechts

25 = (WM_HSLID) Horizontaler Schieber wurde bewegt.
Menu(4) = Handle
Menu(5) = Schieberposition relativ zur umgebenden Box (0-1000).

26 = (WM_VSLID) Vertikaler Schieber wurde bewegt
Menu(4) = Handle
Menu(5) = Schieberposition relativ zur umgebenden Box (0-1000).

27 = (WM_SIZED) Grössenfeld rechts unten wurde gewählt.
Menu(4) = Handle
Menu(5) = alte Fenster-X-Koordinate
Menu(6) = alte Fenster-Y-Koordinate
Menu(7) = neue Fensterbreite
Menu(8) = neue Fensterbreite

28 = (WM_MOVED) grauer Kopfbalken wurde gewählt.
Menu(4) = Handle
Menu(5) = neue Fenster-X-Koordinate
Menu(6) = neue Fenster-Y-Koordinate
Menu(7) = alte Fensterbreite
Menu(8) = alte Fensterhöhe

Menu(4) = Handle oder Menütitel-Index
Menu(5) = X-Koordinate oder Schieberposition
Menu(6) = Y-Koordinate
Menu(7) = Breite
Menu(8) = Höhe
Menu(9) = Eventflag

32 = Maustasten-Ereignis
34 = Kein Ereignis
35 = Tastatur-Ereignis
48 = Pfeil- oder Scrollbalken-Ereignis

```
50 = Menü-, Eckpunkt-, Kopfbalken- oder Schieber-Ereignis
     Menu(10) = Maus-X-Koordinate zum Zeitpunkt des
     Ereignisses (relativ zur linken oberen Bildschirmecke)
 Menu(11) = Maus-Y-Koordinate zum Zeitpunkt des Ereignisses
            (relativ zur linken oberen Bildschirmecke)
 Menu(12) = Maustasten-Status (1=links;2=rechts;3=beide)
 Menu(13) = "Switch"-Tastenstatus
            Shift-Taste rechts  = 1
            Shift-Taste links   = 2
            Control-Taste       = 4
            Alternate-Taste     = 8
            Es sind Kombinationen aller vier Tasten möglich.
            z.B. "Control" + "Shift"-links = 6
 Menu(14) = Tastatur-Code (16-Bit-Wert)
            High = Scan-Code
            Low  = ASCII-Code
 Menu(15) = Anzahl der Mausklicks zum Zeitpunkt des Ereignis.
```

## MENU — Menüpunkt-Attribute bestimmen

**MENU Menü,Wert**

Es können die Eigenschaften (aktiv, checked) von Menüpunkten bestimmt werden. *MENÜ* gibt den Index des Menüpunktes an, dessen Attribute bestimmt werden sollen (siehe unter *MENU(Index)*). Unter Angabe von *Wert* kann der jeweilige Menüpunkt aktiviert, deaktiviert, eine Wahl-Kennzeichnung gesetzt oder diese wieder aufgehoben werden.

| | |
|---|---|
| Wert = 0 | -> "Checkmark" löschen |
| Wert = 1 | -> "Checkmark" setzen |
| Wert = 2 | -> Menüpunkt deaktivieren |
| Wert = 3 | -> Menüpunkt aktivieren |

Bei Angabe der Menünamen durch den Befehl *MENU Menütext$* sind für den Fall, daß "Checkmarks" (Häckchen) gesetzt werden sollen, vor dem jeweiligen Menünamen zwei Leerzeichen als Platzhalter einzusetzen.

**MENU Stringfeld()**

Es wird ein Stringfeld definiert, welches die gewünschten Texte für die Menütitel und deren Menüpunkte enthält. *Stringfeld()* ist ein eindimensionales Textarray, welches wenigstens soviele Elemente haben muß, wie anschließend an Menütiteln und -punkten definiert werden sollen, plus zusätzlicher 20 Feldelemente zur Aufnahme verschiedener Organisationsstrings.

Das erste Menü hat folgenden Aufbau:

String 1 Erster Menütitel (evtl. Programmname)

String 2 Beliebige Überschrift. Unter diesem Menüpunkt kann eine beliebige Programmfunktion eingeordnet werden. Da es der einzige verwendbare Menüpunkt in diesem Menü ist, eignet er sich am besten zur Ausgabe von Programm-Information.

String 3 Reihe von Minuszeichen (bzw. Bindestrichen) Die Anzahl der Striche bestimmt hier die Menübreite. Da unter diesem Menü anschliessend die evtl. vorhandenen Desk-Accessories aufrufbar sind, sollte hier die maximale Länge der Accessory-Titel berücksichtigt werden.

String 4 bis String 9
Sechs Leerstrings, die als Platzhalter für die Accessories dienen. Es dürfen keine Nullstrings ("") übergeben werden!

String10 Nullstring ("") als End-Kennzeichnung.

An diesen Menüaufbau werden nun die frei benennbaren weiteren Menüs angehängt. Diese sind so aufgebaut, daß sich an den jeweiligen Menütitel die einzelnen Menüpunkt-Bezeichnungen anschließen. Dabei muß, wie beim ersten Menü, ein Nullstring ("") den Abschluß zu jedem einzelnen Menü bilden. Abschließend sind dem Feld noch zwei wei-

tere Nullstrings anzufügen. Wird den einzelnen Menüpunkt-Bezeichnungen ein Minuszeichen, bzw. Bindestrich vorangestellt, werden diese im Menü als nicht aktiv (grau) aufgeführt. Falls "checkmarks" (siehe *MENU*) verwendet werden sollen, sind für diese vor dem entsprechenden Menüpunktnamen zwei Leerzeichen einzufügen.

## MENU KILL — Menüzeile löschen

**MENU KILL**

Das Pulldown-Menü wird deaktiviert. Nach Einsatz dieses Befehls kann keine Menü-Auswahl mehr vorgenommen werden. Der Menüzeilentext wird hierdurch nicht gelöscht. Soll das Menü wieder aktiviert werden, ist erneut der Befehl *ON MENU* zu verwenden. Das Menütext-Array muß dazu nicht neu initialisiert werden.

## MENU OFF — Menütitel invertieren

**MENU OFF**

Stellt einen aktivierten Menütitel wieder im Normalmodus (schwarz auf weiß) dar. *MENU OFF* sollte immer eingesetzt werden, wenn ein Menü geöffnet und ein Menüpunkt angewählt wurde. Wurde kein Punkt gewählt und außerhalb des Menüs ein Mausknopf betätigt, ist diese Maßnahme nicht notwendig.

## ON MENU — Verzweigung zur Ereignisfeststellung

**ON MENU**

Ereignis-Überwachung wird eingeschaltet. Dieser Befehl wird überall dort eingesetzt, wo es darauf ankommt, zu erfahren, ob ein Ereignis (Mausbewegung, Tastatur, Window- und Fensteraktionen usw.) eingetreten ist. Dies sollte an häufig wiederkehrenden Programmstellen (in-

nerhalb von Schleifen) geschehen, da der Mitteilungs-Puffer permanent erneuert wird. Wird nicht unverzüglich auf das eingetretene Ereignis reagiert und der Puffer ausgelesen, können neuere Puffereinträge die Mitteilung überschreiben und somit ein falsches Ergebnis liefern. Wird der Befehl nicht eingesetzt, wird auch keine der mit einem der ON MENU GOSUB-Befehle bestimmten Prozeduren aufgerufen.

## ON MENU GOSUB Verzweigung bei Menüauswahl

**ON MENU GOSUB Procedurename**

Es kann eine Prozedur bestimmt werden, zu welcher das Programm verzweigen soll, wenn einer der aktiven Pulldown-Menüpunkte angewählt wurde. Mit *Procedurename* wird die Prozedur angegeben, zu welcher verzweigt werden soll, wenn der Anwender das Pulldown-Menü geöffnet und einen Menüpunkt angeklickt hat. Über den Befehl *MENU* in der angegebenen Prozedur kann dann der gewählte Menüpunkt ermittelt (siehe *MENU*), und entsprechend auf die jeweilige Auswahl reagiert werden.

## ON MENU I/OBOX GOSUB Mausereignis

**ON MENU IBOX Id,Xp,Yp,Br,Hö GOSUB Procedurename**
**ON MENU OBOX Id,Xp,Yp,Br,Hö GOSUB Procedurename**

Es kann eine Prozedur bestimmt werden, zu welcher das Programm verzweigen soll, wenn der Mauszeiger einen von vier möglichen Bildschirmbereichen entweder betritt oder verläßt. Mit jedem dieser beiden Befehle können jeweils zwei Bildschirmbereiche definiert werden, die nach Einsatz von *ON MENU IBOX/OBOX GOSUB*, vorausgesetzt, die Ereignisüberwachung wurde mit *ON MENU* initialisiert, daraufhin überwacht werden, ob der Mauszeiger sich innerhalb (*IBOX*) oder außerhalb (*OBOX*) des jeweiligen Bereichs befindet. Wurde also mit *ON MENU OBOX GOSUB* ein solcher Bereich bestimmt, wird zu der angegebenen Prozedur verzweigt, wenn der Mauszeiger sich aus

diesem Bereich hinausbewegt hat. Bei *ON MENU IBOX GOSUB* tritt die Verzweigung ein, wenn der Mauszeiger das angegebene Feld betritt. *Procedurname* steht für die jeweilige Prozedur, zu welcher verzweigt werden soll. *Id* bedeutet Identifikator und bestimmt, welches der jeweils zwei unabhängigen Felder überwacht werden soll. In *Xp* und *Yp* wird ein X/Y-Koordinatenpaar angegeben, daß die linke obere Ecke des Bereichs beschreibt. Mit *Br* und *Hö* werden die Breite und die Höhe des Feldes bestimmt.

## ON MENU KEY GOSUB — Tastatur-Ereignis

**ON MENU KEY GOSUB Procedurename**

Es kann eine Prozedur bestimmt werden, zu welcher das Programm verzweigen soll, wenn ein Tastatur-Ereignis auftritt. Wurde durch Einsatz des Befehls *ON MENU* bestimmt, daß bei Menu-, oder Window-Ereignissen verzweigt werden soll, und wurde durch *ON MENU KEY GOSUB* eine Prozedur bestimmt, kann in dieser Prozedur der Ereignis-Puffer ausgelesen werden (siehe *MENU*). Unter *MENU(14)* steht dann der Scancode der jeweils vom Anwender gewählten Taste, bzw. Tastenkombination. *MENU(14)* liefert dabei einen Zwei-Byte-Wert, der, um den ASCII-Code der Taste ermitteln zu können, in seine zwei einzelnen Bytes gesplittet werden muß.

## ON MENU MESSAGE GOSUB — Multi-Ereignis

**ON MENU MESSAGE GOSUB Procedurename**

Es kann eine Prozedur bestimmt werden, zu welcher das Programm verzweigen soll, wenn ein Ereignis im Mitteilungs-Puffer gemeldet wurde. Wurde durch Einsatz des Befehls *On Menu* bestimmt, daß bei Menü- oder Window-Ereignissen verzweigt werden soll, und wurde durch *ON MENU MESSAGE GOSUB* eine Prozedur bestimmt, kann in dieser Prozedur der Ereignispuffer ausgelesen (siehe *MENU*) und daraufhin konkret auf den jeweiligen Anwendereinfluß reagiert werden.

## GEMSYS — AES-Routinen aufrufen

**GEMSYS**
(letzten Opcode wiederholen)

**GEMSYS Opcode**
(neuer Opcode)

AES-Systemroutinen können aufgerufen werden. Das AES (Application Environment System) unterstützt grafikorientierte Arbeiten (Windows, Menüs, Alertboxen). *Opcode* enthält die Funktionsnummer der jeweils aufzurufenden GEM-Routine. Vor Aufruf sind die entsprechenden GEM-Parameter-Arrays vorzubereiten. Sollen nur die Array-Inhalte verändert und dieselbe Funktion nochmals aufgerufen werden, kann die Angabe von *Opcode* entfallen. Die jeweils vorzubereitenden Arrays sind unter "Systemdaten und -adressen" mit dem Zusatz *AES* gekennzeichnet.

```
For I=1 To 10
  Dpoke Gintin,620-I*55
  Dpoke Gintin+2,390-I*20
  Dpoke Gintin+4,20+I*20
  Dpoke Gintin+6,10+I*20
  Dpoke Gintin+8,10+I*20
  Dpoke Gintin+10,10+I*20
  Dpoke Gintin+12,620-I*20
  Dpoke Gintin+14,10+I*20
  Gemsys 73
  Gemsys 74
Next I
```

**VDISYS**
(letzten Opcode wiederholen)

**VDISYS Opcode**
(neuer Opcode)

Hiermit VDI-Systemroutinen aufgerufen werden. Das VDI (Virtual Device Interface) unterstützt Grafik- oder Textausgaben, bzw. -einstellungen. *Opcode* enthält die Funktionsnummer der jeweils aufzurufenden GEM-Routine. Vor Aufruf sind die entsprechenden GEM-Parameter-Arrays vorzubereiten. Sollen nur die Array-Inhalte verändert und dieselbe Funktion nochmals aufgerufen werden, kann die Angabe von *Opcode* entfallen. Die jeweils vorzubereitenden Arrays sind unter "Systemdaten und -adressen" mit dem Zusatz *VDI* gekennzeichnet.

```
Dpoke Contrl+2,4
Dpoke Contrl+6,2
Dpoke Contrl+10,3
Dpoke Contrl+12,2
Dpoke Intin,1600
Dpoke Intin+2,2000
Dpoke Ptsin,580
Dpoke Ptsin+2,200
Dpoke Ptsin+12,490
Vdisys 11
For I=1 To 4
  Deffill ,2,I
  Vdisys
Next I
```

**BIOS(Opcode,Parameterliste)**
**XBIOS(Opcode,Parameterliste)**
**GEMDOS(Opcode,Parameterliste)**

BIOS = Basical Input-/Output-System
XBIOS = Extended Basical Input-/Output-System
GEMDOS = Graphic Environment Manager Disk Operating System

Die vom Betriebssystem zur Verfügung gestellten Routinen der einzelnen Untersysteme können aufgerufen werden. Der Atari ST besitzt drei verschiedene Systemebenen, die jede für sich spezielle Aufgaben hat. Über den entsprechenden *Opcode* (Funktionsnummer) lassen sich z.B. Disketten formatieren oder eigene Font-Tabellen laden usw. Die Anzahl der zu übergebenden Parameter ist abhängig von der jeweils aufgerufenen Systemroutine. Werden von der Funktion Longwords erwartet (z.B. bei Adress-Parametern), wird dem entsprechenden Wert ein *L:* vorangesellt. Wird die Angabe dieser Kennung unterlassen, werden die übergebenen Werte als 16-Bit-Words interpretiert. Sollen Strings übergeben werden, ist die jeweilige Startadresse (siehe auch *VARPTR*) anzugeben. Zur Aufnahme eines evtl. Rückgabewertes bzw. der Fehlernummer kann eine beliebige numerische Variable bestimmt werden. Dieser Variablen wird dann das Ergebnis der jeweiligen Funktion zugewiesen. Es handelt sich hier um jene Daten, die nach Aufruf dieser Funktionen im Datenregister D0 eingetragen werden. Dieser Wert kann auch direkt ausgegeben oder in eine Bedingungsabfrage eingebunden werden.

# 13. Systemdaten und -adressen

## ADDRIN AES - Adreß-Inputblock

Startadresse eines Speicherbereichs zur Aufnahme von Adressen, die an eine AES-Funktion übergeben werden sollen.

Falls von der Funktion Adressen benötigt werden, auf welche die Funktion zugreifen soll (z.B. Objektbaumadressen), müssen diese mit *LPOKE* als Integer im ADDRIN-Array (4 Byte je Element) übergeben werden.

## ADDROUT AES - Adreß-Outputblock

Startadresse eines Speicherbereichs, in welchem von der jeweiligen AES-Funktion gegebenenfalls Rückgabeadressen eingetragen werden.

Werden von einer AES-Funktion Adressen zurückgegeben, können diese aus dem ADDROUT-Array (4 Byte je Element) mit *LPEEK* ausgelesen werden.

## CONTRL VDI - Kontrollblock

Startadresse eines Speicherbereichs, in welchen je nach VDI-Funktion die erforderlichen Grundparameter einzutragen sind.

VDI-Initialisierungs-Parameter müßen mit *DPOKE* als Integer im CONTRL-Array (2 Byte je Element) übergeben werden.

| | |
|---|---|
| Contrl | Funktionsnummer ("Opcode") |
| Contrl+2 | Anzahl der Elemente im PTSIN-Array, deren Inhalte von der jeweiligen Funktion übernommen werden sollen. |
| Contrl+4 | Enthält nach Abschluß der Funktion die Anzahl der PTSOUT-Elemente |
| Contrl+6 | Anzahl der Elemente im INTIN-Array, deren Inhalte von der jeweiligen Funktion übernommen werden sollen. |
| Contrl+8 | Enthält nach Abschluß der Funktion die Anzahl der INTOUT-Elemente. |
| Contrl+10 | Funktionsspezifischer Identifikator |
| Contrl+12 | gewähltes Device-Handle |

## DATE$ Systemdatum ermitteln

**DATE$**

Übergibt einen String, der das aktuelle Systemdatum enthält. *DATE$* kann direkt ausgegeben, einer Stringvariablen übergeben, in einen Stringausdruck oder in eine Bedingungsabfrage eingebunden werden. Die Ausgabe erfolgt im Format *dd.mm.yyyy*. Wurden keine Veränderungen des aktuellen Datums vorgenommen, enthält *DATE$* immer das jeweilige TOS-Versionsdatum.

## ERR — Fehlercode ermitteln

**ERR**

Enthält nach Auftreten eines Fehlers eine Identifikationszahl, die dem aufgetretenen Fehler entspricht. *ERR* ist eine reservierte Variable, in der eine Nummer zwischen -128 und 127 eingetragen wird, falls ein Fehler auftritt. *ERR* kann direkt ausgegeben, einer numerischen Variablen übergeben oder in Bedingungsabfragen eingebunden werden.

## ERROR [ERR] — Fehler simulieren

**ERROR Fehlernummer**

Simuliert einen Fehler mit der angegebenen *Fehlernummer*. Es wird eine Fehlermeldung ausgegeben, oder, falls *ON ERROR GOSUB* verwendet wurde, zu der dort angegebenen Fehler-Prozedur verzweigt. Nach Ausgabe einer Fehlermeldung wird das Programm abgebrochen und der Interpreter kehrt zum Editor zurück.

## FATAL — Fehlerart ermitteln

**FATAL**

Enthält nach Auftreten eines Fehlers eine Identifikationszahl, die der Art des jeweiligen Fehlers entspricht. Es wird eine Unterscheidung zwischen Normal- und Bomben-Fehlern getroffen. Ist ein Bomben-Fehler aufgetreten (Adresse des zuletzt ausgeführten BASIC-Befehls ist nicht mehr bekannt) wird der Wert -1 zurückgegeben. Bei allen anderen Fehlern wird eine 0 geliefert. Wurde durch *ON ERROR GOSUB* eine Fehlerprozedur benannt, kann der gelieferte Wert dort wahlweise direkt ausgegeben, an eine Variable übergeben oder in eine Bedingungsabfrage eingebunden werden. Nach Ausgabe einer Fehlermeldung kann *FATAL* im Direktmodus erfragt werden.

## FRE — Freien Speicherplatz ermitteln

**FRE(Dummy)**

Ermittelt die Größe des z.Zt. verbleibenden freien Speicherplatzes in Byte. Es wird eine "Garbage Collection" durchgeführt und anschließend die Größe des freien Speichers zurückgegeben. Eine Garbage Collection bewirkt ein "Aufräumen" des Arbeitsspeichers, wobei sämtliche unbenutzten Speicherbereiche gesammelt und gelöscht werden. *Dummy* ist eine beliebiger Integerwert ohne Bedeutung. Der gelieferte Wert kann wahlweise direkt ausgegeben, an eine Variable übergeben oder in eine Bedingungsabfrage eingebunden werden.

## GB — AES - Anfangsadresse

**G.contrl% = Lpeek(Gb)**
**G.intin% = Lpeek(Gb+8)**
**G.intout% = Lpeek(Gb+12)**
**G.addrin% = Lpeek(Gb+16)**
**G.addrout%= Lpeek(Gb+20)**

Startadresse eines Speicherbereichs, in welchem die Anfangsadressen der AES-Arrays abgelegt sind. Die angegebenen Variablen beinhalten dieselben Adressen, die unter den jeweiligen BASIC-Feldbezeichnungen verfügbar sind.

## GCONTRL — AES - Kontrollblock

Startadresse eines Speicherbereichs, in welchen je nach AES-Funktion gegebenenfalls die erforderlichen Grund-Parameter einzutragen sind.

AES-Initialisierungs-Parameter müßen mit *DPOKE* als Integer im GCONTRL-Array (2 Byte je Element) übergeben werden.

| | |
|---|---|
| Gcontrl | Funktionsnummer ("Opcode") |
| Gcontrl+2 | Anzahl der Elemente im GINTIN-Array, deren Inhalte von der jeweiligen Funktion übernommen werden sollen. |
| Gcontrl+4 | Enthält nach Abschluß der Funktion die Anzahl der GINTOUT-Elemente. |
| Gcontrl+6 | Anzahl der Elemente im ADDRIN-Array, deren Inhalte von der jeweiligen Funktion übernommen werden sollen. |
| Gcontrl+8 | Enthält nach Abschluß der Funktion die Anzahl der ADDROUT-Elemente. |

## GINTIN — AES - Integer-Inputblock

Startadresse eines Speicherbereichs zur Aufnahme von funktionsspezifischen Parametern, die an eine AES-Funktion übergeben werden sollen.

Falls von der Funktion Parameter benötigt werden, auf welche die Funktion zugreifen soll, müssen diese mit *DPOKE* als Integer im GINTIN-Array (2 Byte je Element) übergeben werden. Die Anzahl der Elemente und ihre jeweiligen Bedeutungen sind von der aufgerufenen Funktion abhängig.

## GINTOUT AES - Integer-Outputblock

Startadresse eines Speicherbereichs, in welchem von der jeweiligen AES-Funktion gegebenenfalls Informationen zurückgegeben werden

Werden von einer AES-Funktion Daten zurückgegeben, die keine Adressen darstellen, können diese aus dem GINTOUT-Array (2 Byte je Element) mit *DPEEK* ausgelesen werden. Die Anzahl der Elemente und ihre jeweiligen Bedeutungen sind von der aufgerufenen Funktion abhängig.

## INTIN VDI - Integer-Inputblock

Startadresse eines Speicherbereichs zur Aufnahme von funktionsspezifischen Parametern, die an eine VDI-Funktion übergeben werden sollen.

Falls von der Funktion Parameter benötigt werden, auf welche die Funktion zugreifen soll, müssen diese mit *DPOKE* als Integer im INTIN-Array (2 Byte je Element) übergeben werden. Die Anzahl der Elemente und ihre jeweiligen Bedeutungen sind von der aufgerufenen Funktion abhängig.

## INTOUT VDI - Integer-Outputblock

Startadresse eines Speicherbereichs, in welchem von der jeweiligen VDI-Funktion gegebenenfalls Informationen zurückgegeben werden.

Werden von einer VDI-Funktion Daten zurückgegeben, die keine Koordinaten darstellen, können diese aus dem INTOUT-Array (2 Byte je Element) mit *DPEEK* ausgelesen werden. Die Anzahl der Elemente und ihre jeweiligen Bedeutungen sind von der aufgerufenen Funktion abhängig.

## PTSIN VDI - Punkt-Inputblock

Startadresse eines Speicherbereichs zur Aufnahme von funktionsspezifischen Koordinaten, die an eine VDI-Funktion übergeben werden sollen.

Falls von der Funktion Koordinaten benötigt werden, auf welche die Funktion zugreifen soll, müssen diese mit *DPOKE* als Integer im PTSIN-Array (2 Byte je Element) übergeben werden. Die Anzahl der Elemente und ihre jeweiligen Bedeutungen sind von der aufgerufenen Funktion abhängig.

## PTSOUT VDI - Punkt-Outputblock

Startadresse eines Speicherbereichs, in welchem von der jeweiligen VDI-Funktion gegebenenfalls Koordinaten zurückgegeben werden.

Werden von einer VDI-Funktion Koordinaten zurückgegeben, können diese aus dem PTSOUT-Array (2 Byte je Element) mit *DPEEK* ausgelesen werden. Die Anzahl der Elemente und ihre jeweiligen Bedeutungen sind von der aufgerufenen Funktion abhängig.

## SETTIME [SETT] Uhrzeit und Datum einstellen

**SETTIME Zeit$,Datum$**

Stellt die interne Uhr auf ein neues Datum und auf eine neue Uhrzeit ein. Die mit *SETTIME* aktualisierten Datum- und Zeitangaben können von einem Programm aus mit den Befehlen *TIME$* und *DATE$* abgerufen werden und sind auch auf Systemebene (TOS, Kontrollfeld-Accesssory...) verfügbar. Der Befehl kann die Eingaben *Zeit$* und *Datum$* entweder direkt als Strings in Anführungszeichen oder als Stringvariablen erhalten. Es müssen beide Strings übergeben werden. Die Zeitangabe wird mit Doppelpunkten getrennt, die Datumsangabe mit Punkten.

Format: Zeit$ "hh:mm:ss" oder "hhmmss"
Datum$ "dd.mm.yyyy" oder "dd.mm.yy"

Im ZEIT$-String steht *hh* für Stunden, *mm* für Minuten und *ss* für Sekunden, immer zweistellig. Die Trennzeichen (Doppelpunkte) sind optional. Im DATUM$-String steht *dd* für den Tag, *mm* für den Monat und *yyyy* bzw. *yy* für das Jahr. Die Trennzeichen (Punkt) sind nicht optional, die Jahresangabe kann auch zweistellig erfolgen (z.B. 86 für 1986), falls es sich um eine Angabe zwischen 1980 und 2079 handelt. Ist eine Angabe fehlerhaft (z.B. *DATUM$="00.00.86"* oder *ZEIT$="40:70:90"*), wird diese nicht übernommen.

## TIME$ System-Uhrzeit ermitteln

**Time$**

Übergibt einen String, der die aktuelle Systemuhrzeit enthält. *TIME$* kann direkt ausgegeben, an eine Stringvariable übergeben, in einen Stringausdruck oder in Bedingungsabfragen eingebunden werden. Das Ausgabeformat ist *"hh:mm:ss"*. Die Sekunden der Zeitangabe werden in Zweierschritten erhöht.

# 14. Die neuen V2.0-Befehle

Die neue GfA-BASIC-Version bietet eine Reihe von Befehlen, die das Programmieren zum Vergnügen werden lassen - soweit es das mit der ersten Version nicht schon war. So sind umfangreiche Speichermanipulationen außerhalb der vom Interpreter oder durch Variablenformate vorgegebenen Grenzen möglich. Andererseits wird durch einige Befehle der Einsatz des zu erwartenden BASIC-Compilers vorbereitet und wesentlich vereinfacht.

Zusätzlich zu den folgenden neuen BASIC-Befehlen wurden in die Version V2.0 einige Komfort-Erweiterungen aufgenommen.

Dies sind:

1. Ein neuer Operator: " == "
Das doppelte Gleichheitszeichen testet zwei angegebene Werte auf "ungefähr" gleich. Dabei werden die ersten 28 Bit der beiden Werte verglichen. Stimmen sie überein, ist die Bedingung erfüllt.

2. Der FILL-Befehl arbeitet grundsätzlich im Colormodus. D.h., daß auch im Monochrom-Betrieb jede Fläche unabhängig vom Farbwert gefüllt werden kann. Es ist so z.B. möglich, schwarze Flächen mit weiß oder einem Füllmuster zu füllen.

3. Soll das aktuelle Programm mit einer der beiden Editorfunktionen *SAVE* oder *SAVE,A* auf Diskette gespeichert werden und es ist bereits eine ".BAS"- oder ".LST"-Datei mit gleichem Namen vorhanden, wird die Extension der vorhandenen Datei auf ".BAK" geändert. Sie wird also nicht durch die neue Datei überschrieben.

Will man nun diese ".BAK"-Datei in den Speicher laden, kann man dies (bei ehemaligen ".BAS"-Files) entweder durch den Befehl *LOAD Dateiname* im Direktmodus oder durch die LOAD- bzw. MERGE-Funktion im Editormenü tun. Dazu muß dann in der erscheinenden Fileselect-Box

die Indexzeile angeklickt werden. Anschließend wird die Extension in dieser Zeile auf ".BAK" oder ".*" geändert und der graue Kopfbalken angeklickt. Nachdem dies getan ist, kann die Datei wie üblich durch Doppelklick auf die Namenszeile geladen werden.

Da aus der Extension ".BAK" nicht zu erkennen ist, ob es sich um eine ehemalige ".BAS"- oder ".LST"-Datei handelt, kann es zu den Fehlermeldungen "Kein GfA-BASIC V2.0 Programm" oder *MERGE* - "kein ASCII-File" kommen. Danach weiß man, daß man die jeweils andere Ladefunktion zu verwenden hat.

4. Der Editor behält die zuletzt eingestellte Pfadbezeichnung (z.B. "\Ordner\Unterordner") bei. Wenn man also wieder in das Haupt-Directory möchte, muß man bei der nächsten Editor-Diskettenoperation in der Fileselect-Box das kleine CLOSE-Feld neben dem Kopfbalken anklicken.

5. Im Direktmodus kann die jeweils zuletzt ausgeführte Kommandozeile durch Betätigung der <UNDO>-Taste wieder aufgerufen werden. Dies funktioniert auch, wenn man zum Editor wechselt und erneut die Kommandoebene aufruft.

6. Der INP-Befehl wurde dem ST-BASIC-INP angepaßt. Es werden also jetzt exakt dieselben Tastaturcodes geliefert, wie man es vom ST-BASIC gewohnt war. Die Konvertierung von alten ST-BASIC-Programmen in das GfA-BASIC wird dadurch wesentlich erleichtert. Zumal in Kürze ein Public-Domain-Programm der Firma GfA erscheinen wird, das fast 100prozentig selbständig diese Konvertierung vornimmt.

7. Im Gegensatz zur ersten Version werden die Zahlen 0 bis 3 nicht mehr als Zwei-/Vier- oder Sechs-Byte-Werte intern verarbeitet, sondern durch Ein-Byte-Codes repräsentiert. D.h., daß bei häufiger Verwendung von Flags in diesem Wertebereich (z.B. auch bei der Rückgabevariablen der ALERT-Box) Speicherplatz gespart wird.

".BAS"-Programme der ersten Version können nachträglich verkürzt werden, indem man sie mit *SAVE,A* auf Diskette speichert, danach mit *MERGE* wieder in den gelöschten (*NEW*) Programmspeicher lädt und anschließend wieder mit *SAVE* als ".BAS"-File der Version V2.0 auf Diskette sichert.

8. Mit folgendem Mini-Programm kann ein ".BAS"-Programm beim Run-Only-Interpreter angemeldet werden. Es muß nur der Interpreter angeklickt werden und das Programm startet automatisch.

```
Open "U",#1,"GFABASRO.PRG"
Seek #1,30
Print #1,Chr$(0);"Programmname.BAS"
Print #1,Chr$(0)
Close
```

Als *"Programmname"* muß selbstverständlich der Name des jeweiligen ".BAS"-Programms eingegeben werden. Insgesamt darf der Name die Länge von 60 Zeichen nicht überschreiten.

Noch ein kleiner Trick:

Der Interpreter hat die Eigenschaft, sich alle (!) Variablen-, Label- oder Prozedurnamen, die jemals im Programm verwendet wurden, in einer internen Liste zu merken. Diese Liste wird bei ".BAS"-Programmen auch an die Datei übergeben und beim nächsten Laden wieder übernommen.

Die Namen bleiben dort erhalten, auch wenn sie wieder aus dem Programm entfernt oder ersetzt wurden. Da diese Variablenliste nicht beliebig lang sein kann, kann es vorkommen, daß bei großen Programmen aus scheinbar unerfindlichen Gründen außer Feldvariablen keine weiteren Variablen mehr eingeführt werden können. Um nun diese Variablenliste auf den aktuellen Stand zu bringen, muß man den gleichen Ablauf durchführen, wie oben unter Punkt 7 beschrieben wurde (speichern/laden/speichern). Dadurch werden die nicht mehr benötigten Variablennamen aus der Liste entfernt und die Listenplätze sind für neue Variablennamen wieder verfügbar.

## ! Kommentartext

Ermöglicht das Anbinden von Kommentartext an die jeweilige Befehlszeile. Grundsätzlich hat dieses Ausrufungszeichen exakt die gleiche Funktion wie der Befehl *REM*. Der Unterschied besteht darin, daß der *Kommentartext*, durch dieses Symbol vom Befehlstext getrennt, direkt in die Befehlzeile gesetzt werden kann. Bei DATA- und REM-Zeilen ist dies allerdings nicht erlaubt.

```
a$=S.input(24)                    ! verdeckte Eingabe
Print A$                          ! String ausgeben
Procedure S.input(Anzahl%)        ! Eingabe-Routine
  Dpoke Contrl+2,0                ! keine Ptsins
  Dpoke Contrl+6,2                ! zwei Intins
  Dpoke Intin,4                   ! Tastatur-Eingabe
  Dpoke Intin+2,1                 ! Request-Mode
  Vdisys 33                       ! Set_Mode aufrufen
  Dpoke Intin,Anzahl%             ! Anzahl der Zeichen
  Vdisys 31                       ! Input aufrufen
  For I=0 To Dpeek(Contrl+8)      ! so oft, wie Zeichen
    A$=A$+Chr$(Peek(Intout+1+I*2)) ! String bilden
  Next I                          ! Schleifenende
Return                            ! Zurück zum Programm
```

Diese kleine Routine erlaubt die verdeckte Eingabe eines beliebig langen (*Anzahl%*) Strings. Prinzipiell ist sie identisch mit *INPUT$*. Die Eingabe wird entweder vor Erreichen der max. Länge durch <RETURN> oder durch Erreichen der max. Zeichenanzahl beendet. Der String wird anschließend abhängig von der tatsächlich eingegebenen Anzahl an Zeichen in *A$* zusammengesetzt.

***Var**

Bei Angabe eines Variablennamens mit vorangestelltem Stern wird die Adresse der betreffenden Variablen übergeben. Wird ein Variablenname *Var* beliebigen Typs auf diese Art als Zeiger gekennzeichnet, wird nicht der Variableninhalt, sondern die Variablenadresse übergeben. Bei Feldern und Strings ist dies der zugehörige Descriptor, bei numerischen Variablen die Adresse, an welcher der Variableninhalt zu finden ist. Bei verschiedenen Notwendigkeiten hat man so die Möglichkeit, die Variablen indirekt zu adressieren (vgl. *VARPTR/ARRPTR*).

Der Zeiger kann direkt ausgegeben, einer Variablen übergeben, in einen numerischen Ausdruck oder in Bedingungsabfragen eingebunden werden.

## BASEPAGE — BASIC-Basepage ermitteln

**BASEPAGE**

Es kann die Basepage des GfA-BASIC-Interpreters ermittelt werden. Jedem compilierten oder assemblierten Programm, das vom Desktop oder durch *EXEC* gestartet wird, wird vom Betriebssystem ein Basis-Informations-Block (Basepage) von 256 Byte zugewiesen, in welchem in den ersten 128 Byte grundlegende Daten zum Betrieb dieses Programms eingetragen werden. Die übrigen 128 Byte können auf verschiedene Arten verwendet werden. Z.B. wird hier (falls vorhanden) die Kommandozeile des Programms abgelegt, oder es kann hier der Disk-Transfer-Buffer (GEMDOS-Routine $1A) eingerichtet werden. Die gelieferte Adresse kann direkt ausgegeben, einer Variablen übergeben, in einen numerischen Ausdruck oder in Bedingungsabfragen eingebunden werden.

**BITBLT Q.db%(),Z.db%(),R.db%()**

Es wird ein Zielraster in verschiedenen Modi (AND, OR, XOR etc.) mit einem Quellraster maskiert, oder modifiziert. Dies ist eine Direkt-Zugriffsmöglichkeit auf eine komplizierte VDI-Funktion (COPY RASTER). Die gebräuchlichste Verwendung ist das Verschieben von Bildschirmbereichen auf dem Bildschirm oder in den Speicher (siehe *GET/PUT*). Dabei kann bestimmt werden, in welcher Art das copierte Raster sich auf den Zielbereich auswirken soll (siehe *PUT*). Es können allerdings auch beliebige Speicherbereiche auf diese Art in einem Raster definiert und verknüpft werden. Es sind drei Integerfelder vorzubereiten. Das erste beschreibt in 6 Parametern das Quellraster (Q.db%=Quell-Definitionsblock), während das zweite Feld die gleichen 6 Parameter für das Zielraster enthält (Z.db%=Ziel-Definitionsblock). Durch 9 Parameter des dritten Feldes (R.db%=Rechteck-Definitionsblock) können die beiden Rechtecke (Bildschirmbereich oder idealisiertes Speicher-Rechteck) in Punkten bzw. Bits bestimmt werden.

```
Q.db%(0) = Rasteradresse
           (muß eine gerade Zahl sein).
     (1) = Rasterbreite in Punkten/Bits
           (muß durch 16 teilbar sein).
     (2) = Rasterhöhe in Punkten/Wordzeilen
           (Anzahl der Zeilen mit der oben
           definierten Breite, die quasi
           "untereinandergelegt" die Höhe
           des Rasterblocks ergeben)
     (3) = Rasterbreite in Words
           (immer Rasterbreite in Punkten/16)
     (4) = Rasterformat
           (immer 0, da immer gerätespezifisch)
     (5) = Anzahl der Rasterebenen (Bitplanes)
           (640/400=1 ; 640/200=2 ; 320/200=4)

Z.db%()  = Zielraster (Bedeutungen wie oben)
```

```
R.db%(0) = X-Koordinate der linken, oberen Ecke
           des Quellrechtecks (in Punkten/Bits)
     (1) = Y-Koordinate der linken, oberen Ecke
           des Quellrechtecks (in Punkten/Wordzeilen)
     (2) = X-Koordinate der rechten, unteren Ecke
           des Quellrechtecks (in Punkten/Bits)
     (3) = Y-Koordinate der rechten, unteren Ecke
           des Quellrechtecks (in Punkten/Wordzeilen)

(4)bis(7)= entsprechende Angabe zum Zielrechteck.

     (8) = Verknüpfungsmodus (siehe PUT)
```

Die beiden Rechtecke sollten gleiche Größe haben, um ein nachvollziehbares Ergebnis zu liefern. Werden die Raster in unterschiedlichen Größen definiert, wird immer in der Größe des Quellrasters verknüpft (Vorsicht!).

Zum Experimentieren ist es ratsam, sich auf Operationen innerhalb des Bildschirms oder festgelegte Speicherbereiche zu beschränken, da es nicht leicht ist, die Ergebnisse dieser Rasterfunktion vorherzusagen und somit Fehlfunktioen oder Abstürze des Systems auftreten können.

## BGET — Teildatei in Speicher laden

**BGET #Kanal,Adresse,Anzahl**

Einlesen einer Teildatei an eine bestimmte Speicheradresse. *Kanal* ist der Identifikator einer mit *OPEN* geöffneten Datei. Von der Filepointerposition an werden nun soviel Bytes dieser Datei, wie in *Anzahl* angegeben, in den Speicher gelesen, beginnend mit *Adresse*.

## BMOVE — Speicherbereich bewegen

**BMOVE Quelladresse,Zieladresse,Anzahl**

Es wird ein beliebiger Speicherbereich in einen anderen Speicherbereich kopiert. Dieser sehr wichtige und schnelle Befehl ermöglicht Speichermanipulationen größeren Maßstabs. Im Zusammenhang mit dem Befehl *SPUT* können so z.B. schnelle Bildfolgen oder Bildscrolling erreicht werden. Der mit *Quelladresse* beginnende Speicherbereich wird in der mit *Anzahl* angegebenen Bytegröße an den mit *Zieladresse* beginnenden Speicherbereich kopiert. Der Quellbereich bleibt dabei unverändert.

## BPUT — Speicherbereich in Teildatei speichern

**BPUT #Kanal,Adresse,Anzahl**

Ein definierter Speicherbereich kann in eine bestehende Datei eingefügt werden. Dieser Befehl bildet die Umkehrfunktion zu BGET. Es wird ein Speicherbereich, beginnend mit Adresse und in der Bytegröße Anzahl ab Filepointerposition in eine mit Open geöffnete Datei mit dem Identifikator Kanal geschrieben. Im Gegensatz zu *BSAVE* können hier auch Teile der Datei überschrieben oder angefügt werden.

## CLR — Einzelvariablen löschen

**CLR Var, Var%, Var$, ...**

Löscht einzelne Variablen (außer Feldvariablen). Es wird der Name der zu löschenden Variablen, bzw. eine Liste mehrerer zu löschender Variablen übergeben. Dieser Befehl ersetzt die entsprechenden Null- bzw. Leerstring-Zuweisungen.

## CRSCOL Bildschirm-Cursorspalte ermitteln

**CRSCOL**

Es wird die aktuelle bildschirmbezogene Cursorspalte zurückgegeben. Im Gegensatz zur Funktion *POS*, die die stringbezogene Cursorspalte liefert, kann mit diesem Befehl die Spaltenposition des Cursors auf dem Bildschirm (1-80) ermittelt werden. In Verbindung mit *CRSLIN* bildet diese Funktion das Gegenstück zu *PRINT AT(S,Z)*. Der gelieferte Wert kann direkt ausgegeben, einer Variablen übergeben, in einen numerischen Ausdruck oder in Bedingungsabfragen eingebunden werden.

## CRSLIN Bildschirm-Cursorzeile ermitteln

**CRSLIN**

Es wird die aktuelle bildschirmbezogene Cursorzeile zurückgegeben. Der TOS-Bildschirm besitzt im 8*16-Font ein Raster von 80 Zeichenspalten und 25 Zeichenzeilen. Hiermit kann die Bildschirmzeile ermittelt werden, in welcher sich der Cursor aktuell befindet (1-25). In Verbindung mit *CRSCOL* bildet diese Funktion das Gegenstück zu *PRINT AT(S,Z)*. Der gelieferte Wert kann direkt ausgegeben, einer Variablen übergeben, in einen numerischen Ausdruck oder in Bedingungsabfragen eingebunden werden.

## DEFNUM Ausgabewerte runden

**DEFNUM Stelle**

Alle nach diesem Befehl auszugebenden Werte werden auf die angegebene Stelle gerundet. Es wird die Stelle (3 bis 11) angegeben, auf die alle nachfolgenden Stellen gerundet werden sollen. Dabei wird bei Realzahlen der Vorkommaanteil, der ggf. hinter der definierten Rundungsstelle liegt, als Nullen ausgegeben. Liegt die Stelle im Nachkommabereich, werden alle dahinterliegenden Nachkommastelen

ignoriert. Die Rundung erfolgt mathematisch exakt (*INT(A+0.5)*). Werden Variableninhalte auf diese Art ausgegeben, bleibt der Variableninhalt von der Rundung unberührt. Die Stellendefinition bezieht sich auf alle numerischen Ausgaben und wird erst durch die nächste Definition verändert bzw. abgeschaltet (*DEFNUM 11*).

## EVEN — auf gerade Zahl testen

**EVEN(Zahl)**

Die übergebene Zahl wird darauf getestet, ob sie gerade ist. Es wird der Wert -1 (wahr) zurückgegeben, wenn die Zahl gerade ist. Ist sie das nicht, erhält man den Wert 0 (unwahr). Dieser Wert kann direkt ausgegeben oder in Bedingungsabfragen eingebunden werden.

## EXEC — .PRG/.TOS - Programm laden und starten

**EXEC Modus,"Prg","Kom","Env"**
(als Befehl)

**A=EXEC(Modus,"Prg","Kom","Env")**
(als Funktion)

.PRG /.TOS -Programme werden entweder geladen und gestartet oder nur geladen oder geladen und seine Basepage übergeben. Es können fast alle compilierten oder assemblierten Programme der Art *.PRG* oder *.TOS* geladen werden. Dazu ist ein genügend großer Speicherbereich vorzubereiten (siehe *RESERVE*). *Modus* enthält entweder den Wert 0 (laden und starten) oder 3 (laden). Mit *"Prg"* wird der vollständige Programmname (incl. Pfad) angegeben. *"Kom"* enthält eine evtl. zu übergebende Kommandozeile. Diese Kommandozeile kann dem Programm übergeben werden, um verschiedene Situationen auszulösen. In den meisten Fällen genügt die Angabe eines Leerstrings. *"Env"* ist ein String, der die Programmumgebung beschreibt. Dieser String ist von ATARI nicht genügend dokumentiert, um seine Verwendungsmöglichkeiten gültig beurteilen zu

können. Er kann in jedem Fall als Leerstring angegeben werden.

Wird *EXEC* als Funktion verwendet, wird in der Zuweisungs-Variablen bei Modus = 0 ein evtl. vom aufgerufenen Programm gelieferter Wert aufgenommen. Bei Modus = 3 wird nach dem Ladeprozess in der Variablen die Adresse der Basepage des betreffenden Programms abgelegt.

Nicht ausführbar sind z.B. speicherresidente Programme ("SID"), der SEKA-Assembler oder alle Programme, die in irgendeiner Form Batch-Dateien verwenden.

```
@Executer
Procedure Executer
  Fileselect "*.*","",Selected$
  If Selected$>"" And Selected$<>"\"
    If Exist(Selected$)<>0
      S.pace=Fre(0)
      Reserve 50000
      Alert 2,"1. Laden u. starten|2. Laden u.
         Basepage liefern",1," 1. | 2. ",Dummy%
      If Dummy%=1
        Exec 0,Selected$,"",""
        Reserve S.pace-1000
      Else
        A%=Exec(3,Selected$,"","")
        Print A%
        Alert 2,"Speicher wieder freigeben ?",
            1," Ja |Nein",Dummy2%
        If Dummy2%=1
          Void Gemdos(&H49,L:A%)
          Reserve S.pace-1000
        Endif
      Endif
    Endif
  Endif
Return
```

## FALSE — Unwahr-Konstante (0)

**FALSE**

Enthält den konstanten Wert 0. Bei verschiedenen Funktionen (z.B. *EXIST*) wird ein Wahrheitswert zurückgegeben. Zur besseren Übersichtlichkeit kann bei Bedingungsabfragen oder Zuweisungen statt des Wertes 0 die Konstante *FALSE* verwendet werden (siehe auch *TRUE*).

## FORM INPUT AS — String zur Edition ausgeben

**FORM INPUT Anzahl AS Var$**

Gibt den Inhalt einer Stringvariablen auf dem Bildschirm aus und ermöglicht die nachträgliche Edition dieses Strings oder eines Teils davon. *Anzahl* enthält die Anzahl der Zeichen des mit *VAR$* angegebenen Strings (ab Anfang), die zur Edition ausgegeben werden sollen. Nach erneuter Eingabe und Bestätigung durch <RETURN> wird der neu entstandene String in die Variable übernommen. Der vorherige Inhalt wird komplett ersetzt. Wird eine Leervariable angegeben, erhält dieser Befehl die gleiche Funktion wie *FORM INPUT Anzahl,VAR$*.

## HIDEM — Mauszeiger ausschalten

**HIDEM**

Schaltet den Mauszeiger wirksam ab. Beim Einlesen von Bildschirminhalten durch *BLOAD* direkt in den Bildschirmspeicher, wird das geladenen Bild durch die Mauszeigermaske beschädigt. Dies geschieht auch, wenn der Mauszeiger oberflächlich durch *DEFMOUSE String$ (74,MKI$(0))* unsichtbar gemacht wird.

*HIDEM* schaltet den Mauszeiger zwar ab, die Koordinatenermittlung kann jedoch trotzdem mit *MOUSE X,Y,K* oder *MOUSEX/MOUSEY* durchgeführt werden. Wird bei ausgeschaltetem Mauszeiger eine Fileselect-Box oder Alert-Box ausgegeben, wird der Mauszeiger für die Dauer der Boxbedienung sichtbar. Anschließend wird er wieder gelöscht. Er erscheint auch wieder, ohne neu aufgerufen zu werden (*SHOWM*), wenn das Programm beendet wird. Dieser Befehl ist die Umkehrfunktion zu *SHOWM*.

## HIMEM — erstes Byte hinter BASIC-Bereich

**HIMEM**

Liefert die erste Adresse, die hinter dem vom GfA-BASIC belegt Speicherbereich (Interpreter, Programm, FRE(0)-Bereich und Variablen) liegt. Dieser Befehl ist vor allem dazu notwendig, um bei Verwendung des RESERVE-Befehls die Startadresse des unbenutzten Bereiches zu erhalten (Blockmanipulationen durch *BMOVE*, *BGET*, *BLOAD* etc.).

## INP? — Port-Eingabestatus ermitteln

**INP?(Port)**

Teilt mit, ob am angegebenen Port (siehe *INP*) ein Byte gelesen werden kann. Mit *Port* wird die Kennziffer des Ports angegeben, dessen Status ermittelt werden soll.

| | | |
|---|---|---|
| 0 = | LST: | Drucker |
| 1 = | AUX: | serielle Schnittstelle |
| 2 = | CON: | Tastatur und Bildschirm |
| 3 = | MID: | MIDI In |

Kann ein Byte gelesen werden, wird der Wert -1 (True) zurückgegeben. Andernfalls der Wert 0 (False). Dieser Wert kann wahlweise direkt ausgegeben, einer Variablen übergeben, in einen numerischen Ausdruck oder eine Bedingungsabfrage eingebunden werden.

## MID$() Teilstring ersetzen

**MID$(Var$,Start,Anzahl)=Stringausdruck**

Durch Angabe eines Stringausdrucks kann ein Teil einer bestehenden Stringvariablen ersetzt werden. Mit *VAR$* wird der Name der Stringvariablen angegeben, deren Inhalt verändert werden soll. *Start* enthält die Stelle im alten String, ab welcher der neue String eingesetzt werden soll. *Anzahl* kann wahlweise verwendet werden und gibt ggfls. die Anzahl der Zeichen an, die ab *Start* maximal ersetzt werden sollen. Wird die Angabe von *Anzahl* unterlassen, werden entweder soviele Zeichen ersetzt, wie *Stringausdruck* enthält, oder es werden, wenn *Stringausdruck* länger ist, als die ab *Start* verbleibende Stringlänge, die Restzeichen bis zum Ende des Strings ersetzt. Die ursprüngliche Stringlänge wird durch diese Operation nicht verändert.

## MONITOR Maschinen-Monitor aufrufen

**MONITOR(Parameter)**

Wurde das GfA-BASIC von einem residenten Maschinenmonitor aus gestartet, kann dieser hiermit aufgerufen werden. In *Parameter* hat man die Möglichkeit, optional Werte an den Monitor zu übergeben (z.B. Adressen). Es können hier nur Monitore bzw. Debugger verwendet werden, die bestimmte Bedingungen erfüllen. Hierzu gehört z.B. der "SID" des Entwicklungspakets von Digital Research.

Die näheren Umstände ausreichend zu erläutern, würde in diesem Fall zu weit führen. Versuche des Autors, auf diese Art mit dem "SID" zu kooperieren, blieben vorerst erfolglos.

## ODD — auf ungerade Zahl testen

**ODD(Zahl)**

Die übergebene Zahl wird darauf getestet, ob sie ungerade ist. Es wird der Wert -1 (wahr) zurückgegeben, wenn die Zahl ungerade ist. Ist sie das nicht, erhält man den Wert 0 (unwahr). Dieser Wert kann direkt ausgegeben oder in Bedingungsabfragen eingebunden werden.

## ON MENU BUTTON GOSUB — Mausereignis

**ON MENU BUTTON Anzahl,Taste,Status GOSUB Procedure**

Verzweigung zu einer Prozedur, wenn eine vorgegebene Maustaste mehrmals bedient wird (Doppel-/Dreifach-Klicks). Mit *Anzahl* wird die Anzahl der maximalen Klicks angegeben, die berücksichtigt werden sollen. Dies hat kein Ausstiegsfunktion, sondern bestimmt nur, daß bis zu einer bestimmten Klickanzahl die Klicks überhaupt registriert werden. *Taste* gibt an, auf welche Tastenbetätigung reagiert werden soll.

| | |
|---|---|
| 0 = | keine Taste |
| 1 = | linke Maustaste |
| 2 = | rechte Maustaste |
| 3 = | beide Maustasten zugleich |

*Status* bestimmt den Status der gewählten Taste, der zum Ereignis führen soll.

| | |
|---|---|
| 0 = | keine Taste gedrückt |
| 1 = | linke Taste gedrückt |
| 2 = | rechte Taste gedrückt |
| 3 = | beide Tasten gleichzeitig gedrückt |

Da die Richtigkeit der obigen Angaben nicht belegt werden konnte und außerdem die jeweilige Analyse etwas kompliziert ist, werden hier verschiedene Möglichkeiten aufgeführt, die wohl den Bedarf an Klick-Varianten decken

dürften. Es werden hier immer nur maximal Zweifach-Klicks berücksichtigt, da hier auch die Grenze des Machbaren liegen dürfte. Dreifach-Klicks sind schon etwas für Fingerkünstler.

**ON MENU BUTTON 2,0,1 GOSUB Button**
Es wird grundsätzlich verzweigt und auf alle Tastenkombinationen reagiert.

**ON MENU BUTTON 2,1,1 GOSUB Button**
Es wird nur verzweigt, wenn beide Tasten oder linke Taste gedrückt werden. Reagiert nur auf linke Taste oder beide Tasten gleichzeitig.

**ON MENU BUTTON 2,2,1 GOSUB Button**
Es wird immer verzweigt, wenn nicht die rechte Taste und nicht beide Tasten gleichzeitig gedrückt werden. Reagiert nur auf Taste 1.

**ON MENU BUTTON 2,3,1 GOSUB Button**
Es wird nur verzweigt, wenn linke Taste gedrückt wird. Reagiert nur auf linke Taste.

**ON MENU BUTTON 2,2,3 GOSUB Button**
Es wird nur verzweigt, wenn die rechte Taste oder beide Tasten zugleich gedrückt werden. Reagiert nur auf rechte Taste oder beide Tasten zugleich.

**ON MENU BUTTON 2,3,2 GOSUB Button**
Es wird nur verzweigt, wenn die rechte Taste gedrückt wird. Reagiert nur auf rechte Taste.

**ON MENU BUTTON 2,3,3 GOSUB Button**
Es wird nur verzweigt, wenn beide Tasten gleichzeitig gedrückt werden. Reagiert nur auf beide Tasten zugleich.

Die Bedingungen, die mit dem Ereignis verknüpft werden sollen, sind in der angegebenen Pozedur zu stellen. Dazu kann aus *Menu(15)* die Anzahl der tatsächlich eingetretenen Mausklicks ausgelesen werden. Die X/Y-Koordinaten der Maus zum Zeitpunkt des Ereignisses können aus *Menu(10)*" für X, bzw. *Menu(11)* für Y bezogen werden.

## OPTION — Compiler-Befehle übergeben

**OPTION "Befehlstext"**

Dem GfA-BASIC-Compiler können Arbeitsanweisungen übermittelt werden. Zu diesem Befehl gibt es keinerlei Dokumentation. Mit genaueren Informationen ist erst zum Erscheinen des GfA-Compilers zu rechnen.

## OPTION BASE — Feld-Startindex bestimmen

**OPTION BASE 0**
(Startindex = 0)

**OPTION BASE 1**
(Startindex = 1)

Der Startindex der verwendeten Felder kann gewählt werden (0 oder 1). Da es manchmal etwas kompliziert und auch ungünstig sein kann, in einem Vektor mit Null-Elementen zu arbeiten, hat man hier die Möglichkeit, dieses Null-Element zu eliminieren bzw. es wieder einzurichten. Diese Definition wirkt sich allerdings auf alle Felder, die im Programm Verwendung finden gleichzeitig aus. Es kann jedoch ohne weiteres innerhalb des Programms mehrmals der Startindex geändert werden, da die definierten Elemente sich dem neuen Index anpassen.

Wurde also im Programm mit *OPTION BASE 1* begonnen, die Felder zu füllen und man führt anschließend das Null-Element ein, wird der gesamte Feldinhalt um eine Stelle nach unten versetzt. Im umgekehrten Fall (zuerst mit Null-Elementen) werden bei Änderung auf Basis 1 alle Elemente-Inhalte um eine Stelle nach oben versetzt.

## OUT? Port-Ausgabestatus ermitteln

**OUT?(Port)**

Liefert Information darüber, ob der angegebene Port (siehe *OUT*) ausgabebereit ist. Mit *Port* wird die Kennziffer des Ports angegeben, dessen Status ermittelt werden soll.

| | | |
|---|---|---|
| 0 = | LST: | Drucker |
| 1 = | AUX: | serielle Schnittstelle |
| 2 = | CON: | Tastatur und Bildschirm |
| 3 = | MID: | MIDI In |

Kann ein Byte ausgegeben werden, wird der Wert -1 (*TRUE*) zurückgegeben. Andernfalls der Wert 0 (*FALSE*). Dieser Wert kann wahlweise direkt ausgegeben, einer Variablen übergeben, in einen numerischen Ausdruck oder eine Bedingungsabfrage eingebunden werden.

## RESERVE BASIC-Speichergröße bestimmen

**RESERVE Anzahl**

Legt die für BASIC verfügbare (*FRE(0)*) Speichergröße fest. Mit *Anzahl* wird die neu einzurichtende Größe des BASIC-Arbeitsspeichers in Byte angegeben. Bei einer Verminderung des BASIC-Speichers kann der damit oberhalb des BASIC-Speichers enstandene freie Speicherbereich nach Belieben verwendet werden. Befehle wie *BMOVE*, *BLOAD* oder *BGET*, aber auch die Verwendung von ".RSC"-Files (falls Resource Construction Set zur Hand) bieten sich dazu an. Wird der BASIC-Speicher vergrößert, ist dabei zu beachten, daß das GEM und GEMDOS zur Abwicklung verschiedener Prozesse (Alert-Box, Fileselect-Box, DTA etc.) auf einen Speicherbereich direkt unterhalb des Bildschirmspeichers (Xbios(2)) zugreift. Der Interpreter setzt die obere Grenze des freien BASIC-Speichers bei Start automatisch 16384 Byte unterhalb des Video-RAMs. Es wäre also möglich, den Speicher bis zum Video-RAM voll auszuschöpfen (*RESERVE FRE(0)+16384*). In diesem Fall wären *HIMEM* und *XBIOS(2)* identisch. Wird so verfahren, hat das GEM

keine Möglichkeit mehr, z.B. die Fileselect-Box darzustellen. Der Speicher sollte deshalb in der maximalen Größe so gewählt werden, daß unterhalb von *XBIOS(2)* noch ein Block von ca. 4000 Byte für das GEM freigehalten wird.

Nutzt man bei Verminderung des BASIC-Speichers den darüberliegenden Bereich bis an die Bildschirmgrenze aus, werden innerhalb dieser 4000 Byte Daten durch das GEM überschrieben (Vorsicht!). Um jeglichen Zugriff des GEM auf den erhaltenen Bereich oberhalb des BASIC zu verhindern, kann ein Speicherbereich außer durch *RESERVE* auch durch die GEMDOS-Funktion $48 (Malloc) reserviert werden. Hiermit reservierter Speicher muß allerdings durch eine weitere GEMDOS-Funktion $49 (*MFREE*) wieder freigegeben werden (siehe Kapitel "Systemroutinen").

Die Freigabe durch die Funktion $49 ist auch notwendig, wenn mit dem Befehl *EXEC 3...* ein Programm in den Speicher geladen wurde, das nicht gestartet wird und somit den belegten Speicher nicht wieder abgibt.

## SGET — Bildschirm speichern

**SGET Var$**

Überträgt den aktuellen Bildschirmspeicher in eine Stringvariable. Mit *VAR$* wird der Name einer Stringvariablen übergeben, die den Inhalt des Bildschirmspeichers aufnehmen soll. Im Gegensatz zum *GET X0,Y0,X1,Y1,A$* muß hier nicht mit geschnittenen Words operiert zu werden. D.h. zum einen, daß dieser Befehl ca. 15 mal schneller ist als *GET*, aber auch, daß der Variableninhalt ein anderes Format besitzt. Ein mit *SGET* gelesener Bildschirm kann deshalb nicht durch *PUT* dirigiert werden. Dieser Befehl bildet das Gegenstück zu *SPUT*.

Er ist funktionell identisch mit:

```
A$=Space$(32000)
Bmove Xbios(2),Varptr(A$),32000
```

## SHOWM — Mauszeiger einschalten

SHOWM

Der durch *HIDEM* unsichtbare Mauszeiger kann entweder durch diesen Befehl, durch *VDISYS 122* oder durch Rückkehr zum Editor wieder sichtbar gemacht werden.

## SPUT — Bildschirm setzen

SPUT Var$

Überträgt den Inhalt einer Stringvariablen an den Anfang des Bildschirmspeichers. *VAR$* ist der Name der Variablen, deren Inhalt in den Bildschirmspeicher übertragen werden soll. Dies muß nicht unbedingt eine durch *SGET* initialisierte Variable sein. Ist der Variableninhalt größer als 32000 Zeichen (Byte), werden nur die ersten 32000 Zeichen (Byte) berücksichtigt. Dieser Befehl bildet das Gegenstück zu *SGET*. Im Vergleich zu *PUT 0,0,A$* ist er ebenfalls erheblich schneller.

## TRUE — Wahr-Konstante (-1)

TRUE

Enthält den konstanten Wert -1. Bei verschiedenen Funktionen (z.B. *EXIST*) wird ein Wahrheitswert zurückgegeben. Zur besseren Übersichtlichkeit kann bei Bedingungsabfragen oder Zuweisungen statt des Wertes -1 die Konstante *TRUE* verwendet werden (siehe auch *FALSE*).

## TYPE — Variablen-Typ ermitteln

**TYPE(Zeiger)**

Ermöglicht die Ermittlung des Typs der Variablen, auf welche mit dem angegebenen Pointer gezeigt wird. Mit *Zeiger* wird ein numerischer Pointer angegeben, dessen Inhalt auf die Adresse der Variablen zeigt, deren Typ zu ermitteln ist (siehe *). Um den Typenindex eines bestimmten Variablentyps zu ermitteln, kann auch der mit einem Stern gekennzeichnete Variablenname direkt angegeben werden. Dabei wird von der Funktion ein Wert von -1 bis 7 geliefert.

| | |
|---|---|
| -1 | Fehler aufgetreten |
| 0 | Realzahlvariable (Var) |
| 1 | Stringvariable (Var$) |
| 2 | Integervariable (Var%) |
| 3 | Boolsche Variable (Var!) |
| 4 | Real-Feldvariable (Var()) |
| 5 | String-Feldvariable (Var$()) |
| 6 | Integer-Feldvariable (Var%()) |
| 7 | Boolsche Feldvariable (Var!()) |

Dieser Wert kann wahlweise direkt ausgegeben, einer Variablen übergeben, in einen numerischen Ausdruck oder eine Bedingungsabfrage eingebunden werden.

## VDIBASE — VDI-Feldbasis ermitteln

**VDIBASE**

Bietet die Möglichkeit, die Basisadresse des VDI-Parameterblocks zu ermitteln. Es handelt sich hier um einen Speicherbereich von ca. 300 Byte, den das GEM zur Verwaltung der verschiedensten VDI-Parameter einrichtet. Hier liegen z.B. die Informationen über die eingestellte Schriftgröße, Grafikmodus usw. Es sind allerdings auch Daten dabei, die zur Verwaltung der VDI-Umgebung benötigt werden (Open/Close Workstation, Fonts, Bildschirmorganisation usw.). Da es nun von ATARI keine Do-

kumentation zu diesem Bereich gibt, kann nicht gesagt werden, welche Bytes/Words/Longs dieses Bereiches sich auf die verschiedenen Funktionen auswirken. Wird in diesen Bereich gepoket, können dadurch u.U. die verschiedenen VDI-Parameter beeinflußt, aber auch Fehlfunktionen des Systems ausgelöst werden, die dann zum Absturz führen können. Oberhalb des BASIC-Interpreters werden von diesem verschiedene Tabellen und die Programmvariablen verwaltet. Vdibase schließt sich an diesen Bereich an. Im Anschluß an diesen VDI-Bereich wird vom BASIC-Interpreter eine Variablenliste angelegt, auf die dann wiederum das jeweilige BASIC-Programm folgt.

Dieses Programm listet die verschiedenen Formatwerte des VDI-Blocks auf:

```
Print "BYTES:"
For I=0 To 300
  Print Peek(Vdibase+I)'
Next I
Print Chr$(10);Chr$(13);"WORDS:"
For I=0 To 300 Step 2
  Print Dpeek(Vdibase+I)'
Next I
Print Chr$(10);Chr$(13);"LONGS:"
For I=0 To 300 Step 4
  Print Lpeek(Vdibase+I)'
Next I
Void Inp(2)
```

## VOID — Funktionsaufruf ohne Rückgabe

**VOID Funktion(Parameter)**

Ruft eine Funktion auf, ohne dieser eine Rückgabevariable zur Verfügung zu stellen. In vielen Fällen ist die Übergabe einer Aufnahmevariablen bei Funktionsaufrufen unnötig, da diese Variablen oft als Dummy eingesetzt werden. Um Speicherplatz (und Zeit) zu sparen, können diese Funktionen mit *VOID* aufgerufen werden.

```
statt:  A=Inp(2)          ! auf Taste warten
 == >:  Void Inp(2)

statt:  A=Fre(0)          ! Garbage-Collection
 == >:  Void Fre(0)

statt:  A=Gemdos(&HE,1)   ! Disk B aufrufen
 == >:  Void Gemdos(&HE,1)
```

## VSYNC — Vertikal-Blank abwarten

**VSYNC**

*VSYNC* wartet auf das nächste Vertikal-Blank (Strahlrücklauf/Bildaufbau). Bei Grafikausgaben mit *SPUT* oder *PUT* kann es sinnvoll sein, den nächsten Bildneuaufbau abzuwarten. Durch die Vertikale Synchronisation kann somit das Interferenz-Flimmern eingeschränkt werden. Bei Grafikausgaben, die mehr Zeit in Anspruch nehmen, als der Computer zu einem Bildaufbau benötigt (monochrom = 70 Hertz), treten allerdings auch wieder Interferenzen auf.

## 14.1 Systemroutinen

Das Betriebssystem des ATARI ST bietet ein fast unüberschaubare Fülle an Systemroutinen, die durch verschiedene BASIC-Funktionen genutzt werden können. Die hier aufgeführten Funktionen sind nicht vollständig. Der Autor hat sich erlaubt, alle die Routinen zu vernachlässigen, die durch entsprechende BASIC-Befehle auf Interpreter-Ebene verfügbar sind. (z.B. *DFREE(N)* für *GEMDOS &H36 <Get DiskFreeSpace>*, *MKDIR* für *GEMDOS &H39 <MKDIR>*, die AES-WIND_-Funktionen usw.). Einige andere Funktionen wurden deshalb nicht erläutert, da sie derart komplex sind, daß ihre Beschreibung den Rahmen des Buches sprengen würde (z.B. *XBIOS 18 <protopt>*, die AES-OBJC_-Funktionen usw.). Wer sich näher für diese Themen interessiert, wird nicht darum herumkommen, sich das Buch "ATARI ST INTERN" (Brückmann/Englisch/Gerits) oder "Das große GEM-Buch zum ATARI ST" (Szczepanowski/Günther) zu kaufen. Beide Bücher sind im DATA BECKER Verlag erschienen, sehr ausführlich und im Handel erhältlich.

Die in der Kopfzeile links angeführten Bezeichnungen sind reine Funktionsnamen, wie sie von DIGITAL RESEARCH eingeführt wurden. Sie haben mit dem Funktionsaufruf nichts zu tun.

## 14.2 AES

Im AES gibt es die Möglichkeit, sich nach Funktionsaufruf Auskunft darüber geben zu lassen, ob die Funktion ordnungsgemäß ausgeführt wurde oder ob ein Fehler aufgetreten ist. Bei Bedarf kann dies in den meisten Fällen nach dem Funktionsaufruf wie folgt ermittelt werden:

R.eturn%=Dpeek(Gintout)

Wird in *R.eturn* ein positiver Wert geliefert, ist kein Fehler aufgetreten. Eine Null bedeutet Fehler. Es gibt jedoch nicht jede Funktion diese Auskunft.

### GRAF_HANDLE — Applikationshandle ermitteln

Übergibt Handle der anfragenden Applikation und jeweilige Zeichengröße.

```
Gemsys 77
Handle%=Dpeek(Gintout)
Zeichenbreite%=Dpeek(Gintout+2)
Zeichenhoehe%=Dpeek(Gintout+4)
Zeichenboxbreite%=Dpeek(Gintout+6)
Zeichenboxhoehe%=Dpeek(Gintout+8)
```

### APPL_READ — Ereignispuffer lesen

Aus dem Applikationspuffer wird eine vorgegebene Anzahl Bytes gelesen.

```
Dpoke Gintin,Appl.Kennziffer
Dpoke Gintin+2,Byteanzahl
Lpoke Addrin,Pufferadresse
Gemsys 11
```

## APPL_WRITE — Ereignispuffer schreiben

In den Applikationspuffer wird eine vorgegebene Anzahl Bytes geschrieben.

```
Dpoke Gintin,Kennz. der empfangenden Applikation
Dpoke Gintin+2,Anzahl zu schreibender Bytes
Lpoke Addrin,Adresse des Zeichenpuffers
Gemsys 12
```

## RSRC_LOAD — ".RSC"-File laden

Reserviert Speicher und lädt eine Resource-Datei in diesen Bereich.

```
A$="Name der Resource-Datei"+Chr$(0)
Lpoke Addrin,Varptr(A$)
Gemsys 110
```

## RSRC_FREE — ".RSC"-Speicher freigeben

Der zuletzt durch RSRC_LOAD belegte Speicher wird freigegeben und der geladene Objektbaum gelöscht.

```
Gemsys 111
```

## RSRC_GADDR — Objekt-Adresse ermitteln

Es kann die Adresse des zuletzt geladenen Baumes oder einer Objektstruktur ermittelt werden.

```
Dpoke Gintin,0  (= Baum) oder Strukturtyp
Dpoke Gintin+2,Objektindex
Gemsys 112
Adresse%=Lpeek(Addrout)
```

## FORM_DO

AES-Kontrolle

AES überwacht die Formulareingaben und registriert Abbruchbedingungen.

```
Dpoke Gintin,0  oder Index des ersten Textfeldes
Lpoke Addrin,Objektbaumadresse
Gemsys 50
E.ndobjekt%=Dpeek(Gintout)
```

## FORM_CENTER

Formular zentrieren

Das aufgerufene Formular wird in der Mitte des Bildschirms dargestellt und die errechneten Koordinaten und Maße zurückgegeben.

```
Lpoke Addrin,Objektbaumadresse
Gemsys 54
X.Koordi%=Dpeek(Gintout+2)
Y.Koordi%=Dpeek(Gintout+4)
F.ormbreite%=Dpeek(Gintout+6)
F.ormhoehe%=Dpeek(Gintout+8)
```

## OBJC_DRAW

Objektbaum zeichen

Der mit *RSRC_LOAD* geladene Objektbaum wird auf dem Bildschirm innerhalb eines Begrenzungsrechtecks bis zu einer bestimmbaren Ebene gezeichnet.

```
Dpoke Gintin,Index des Startobjekts
Dpoke Gintin+2,Anzahl zu zeichnender Ebenen
Dpoke Gintin+4,X-Koord. d. Clip-Box
Dpoke Gintin+6,Y-Koord. d. Clip-Box
Dpoke Gintin+8,Breite d. Clip-Box
Dpoke Gintin+10,Höhe der Clip-Box
Lpoke Addrin,Objektbaumadresse
Gemsys 42
```

## OBJC_OFFSET

Objektkoordinaten ermitteln

Es lassen sich die bildschirmbezogenen Koordinaten (linke, obere Ecke des Bildschirms/Objekts) hiermit feststellen. Die ermittelten Daten können nach Funktionsende aus *X.koo%/Y.koo%* ausgelesen werden.

```
Dpoke Gintin,Index des Einzelobjekts
Lpoke Addrin,Objektbaumadresse
Gemsys 42
X.koo%=Dpeek(Gintout)
Y.koo%=Dpeek(Gintout+2)
```

## MENU_TEXT

Pulldown-Menütext ändern

Der Textinhalt eines Pulldown-Menüpunktes kann im laufenden Programm geändert werden. Der neue String darf nicht länger sein als der alte.

```
A$="Neuer Menüpunkt-Text"+Chr$(0)
Dpoke Gintin,Menüpunktindex
Lpoke Addrin,Menübaumadresse
Lpoke Addrin+4,Varptr(A$)
Gemsys 34
```

## GRAF_RUBBERBOX

Rubberbox aufrufen

Eine Rubberbox ist ein rechteckiger Rahmen, der mit der linken oberen Ecke fixiert ist und dessen rechte untere Ecke dem Mauszeiger folgt. Die Funktion ist bei gedrückter Maustaste aufzurufen. *Letzt.b%* und *Letzt.h%* enthalten nach Ende der Funktion die letzte Breite, bzw. die letzte Höhe der Box in Pixels.

```
Dpoke Gintin,X-Koord. der linken, oberen Ecke
Dpoke Gintin+2,Y-Koord. der linken, oberen Ecke
Dpoke Gintin+4,kleinste Boxbreite
Dpoke Gintin+6,kleinste Boxhöhe
Gemsys 70
Letzt.b%=Dpeek(Gintout+2)
Letzt.h%=Dpeek(Gintout+4)
```

## GRAF_DRAGBOX

Box im Rahmen verschieben

Es wird ein Bildschirmbereich definiert, innerhalb dessen sich eine kleinere Box vorgegebener Größe bewegen läßt. Die Funktion ist bei gedrückter Maustaste aufzurufen. *Letzt.x%* und *Letzt.y%* enthalten nach Funktionsende die X/Y-Koordinaten der beweglichen Box bei Loslassen der Maustaste.

```
Dpoke Gintin,Breite des kleinen Rechtecks
Dpoke Gintin+2,Höhe des kleinen Rechtecks
Dpoke Gintin+4,X-Startkoord. (Mousex)
Dpoke Gintin+6,Y-Startkoord. (Mousey)
Dpoke Gintin+8,X-Koord. der linken, oberen Ecke
               des umfassenden Rechtecks
Dpoke Gintin+10,Y-Koord. der linken, oberen Ecke
                des umfassenden Rechtecks
Dpoke Gintin+12,Breite des umfassenden Rechtecks
Dpoke Gintin+14,Höhe des umfassenden Rechtecks
Gemsys 71
Letzt.x%=Dpeek(Gintout+2)
Letzt.y%=Dpeek(Gintout+4)
```

## GRAF_MOVEBOX

Movebox aufrufen

Es wird ein Rahmen konstanter Größe gezeichnet, der sich in mehreren Sequenzen von einer bestimmbaren Bildschirmposition zu einer anderen bestimmbaren Position bewegt.

```
Dpoke Gintin,Breite der Box
Dpoke Gintin+2,Höhe der Box
Dpoke Gintin+4,X-Startkoordinate
Dpoke Gintin+6,Y-Startkoordinate
Dpoke Gintin+8,X-Zielkoordinate
Dpoke Gintin+10,Y-Zielkoordinate
Gemsys 72
```

## GRAF_GROWBOX

ausdehnendes Rechteck

Es wird eine Box gezeichnet, die sich von einer Bildschirmposition in einer bestimmbaren Größe zu einer anderen Bildschirmposition in einer anderen bestimmbaren Größe ausdehnt.

```
Dpoke Gintin,X-Koord. des Startrechtecks
Dpoke Gintin+2,Y-Koord. des Startrechtecks
Dpoke Gintin+4,Breite des Startrechtecks
Dpoke Gintin+6,Höhe des Startrechtecks
Dpoke Gintin+8,X-Koord. des Zielrechtecks
Dpoke Gintin+10,Y-Koord. des Zielrechtecks
Dpoke Gintin+12,Breite des Zielrechtecks
Dpoke Gintin+14,Höhe des Zielrechtecks
Gemsys 73
```

## GRAF_SHRINKBOX schrumpfendes Rechteck

Es wird eine Box gezeichnet, die sich von einer Bildschirmposition in einer bestimmbaren Größe zu einer anderen Bildschirmposition in einer anderen bestimmbaren Größe zusammenschrumpft. Es sind dieselben 8 Parameter anzugeben, die bei *GRAF_GROWBOX* beschrieben wurden.

```
Gemsys 74
```

## GRAF_MKSTAT Switch-Tastenstatus ermitteln

Es kann festgestellt werden, welche der Wechseltasten CONTROL/SHIFT/ALTERNATE betätigt wurde. Kombinationen werden als Addition der Werte geliefert.

| | |
|---|---|
| Status 1 | rechte Shift-Taste |
| Status 2 | linke Shift-Taste |
| Status 4 | Control-Taste |
| Status 8 | Alternate-Taste |

```
Gemsys 79
Status=Dpeek(Gintout+8)
```

## Set Grafik Text Special Effects — Textart setzen

Erlaubt die Einstellung verschiedener Textarten. Obwohl diese Funktion denselben Zweck erfüllt, wie der 2. Parameter des BASIC-Befehls *DEFTEXT*, soll er hier aufgeführt werden, da man mit Übergabe von *Handle%* 1 auch die Textart des Desktops, der Alert- bzw. Fileselect-Box oder von ".RSC"-Formularen verändern kann. Der in *T.art%* zu übergebende Wert ist identisch mit den Werten, die bei *DEFTEXT* erklärt wurden.

```
Dpoke Contrl+2,0
Dpoke Contrl+6,1
Dpoke Contrl+12,Handle%
Dpoke Intin,T.art%
Vdisys 106
```

## Set Character Cell High — Textgröße setzen

Erlaubt die Einstellung verschiedener Textgrößen. Bei *Handle%* 2 erfüllt diese Funktion ebenfalls denselben Zweck wie der entsprechende DEFTEXT-Parameter. Mit Übergabe von *Handle%* 1 können auch hiermit die verschiedenen AES-Prozesse beeinflußt werden. Der in *T.high%* zu übergebende Wert kann ebenfalls aus den DEFTEXT-Werten ausgewählt werden.

```
Dpoke Contrl+2,0
Dpoke Contrl+6,1
Dpoke Contrl+12,Handle%
Dpoke Intin,T.high%
Vdisys 107
```

## Set Clipping Rectangle

Grafikbereich definieren

Alle auf diese Funktion folgenden Grafikausgaben (*PBOX*, *LINE*, etc.) werden auf den hiermit festgelegten Bildschirmbereich begrenzt. Blockbefehle wie *PUT*, *SPUT* sind davon ausgenommen.

```
Dpoke Contrl+2,4
Dpoke Contrl+6,1
Dpoke Intin,Flag (Ausschnitt an=1 / aus=0)
Dpoke Ptsin,X-Koord. der linken, oberen Ecke
Dpoke Ptsin+2,Y-Koord. der linken, oberen Ecke
Dpoke Ptsin+4,X-Koord. der rechten, unteren Ecke
Dpoke Ptsin+6,Y-Koord. der rechten, unteren Ecke
Vdisys 129
```

## Set Fill Perimeter Visibility

Box-Rahmen

Die Befehle *PBOX*, *PRBOX*, *PELLIPSE*, *PCIRCLE* und *POLYFILL* umranden die dargestellten Füllmusterflächen mit einer Linie in der eingestellten Objektfarbe. Diese Umrahmung kann hiermit aus- oder eingeschaltet werden.

```
Dpoke Intin,Flag (Rahmen aus = 0 / ein = 1)
Vdisys 104
```

## Inquire Color Representation

Farbmischung

Es können die einzelnen Farbanteile (RGB) einer bestimmten oder der aktuellen Objektfarbe ermittelt werden.

```
Dpoke Contrl+2,0
Dpoke Contrl+6,2
Dpoke Intin,Farbindex bzw. Register
Dpoke Intin+2,Flag (aktuelle Farbe=1 / andere=0)
Vdisys 104
Farbindex%=Dpeek(Gintout)
Rotanteil%=Dpeek(Gintout+2)
Gruenanteil%=Dpeek(Gintout+4)
Blauanteil%=Dpeek(Gintout+6)
```

## Current Fill Area Attributs

Grafikattribute

Es werden mehrere aktuelle Grafikeinstellungen zurückgegeben. In den drei Anteilsvariablen werden je nach Anteil der jeweiligen Farbe Werte zwischen 0 und 1000 geliefert.

```
Dpoke Contrl+2,0
Dpoke Contrl+6,0
Vdisys 37
Fuelltyp%=Dpeek(Intout)
Fuellfarbe%=Dpeek(Intout+2)
Fuellmuster%=Dpeek(Intout+4)
Grafmod%=Dpeek(Intout+6)
Rahmen%=Dpeek(Intout+8)
```

## 14.4 GEMDOS

Bei GEMDOS-Aufrufen mit Angabe einer Rückgabevariablen kann nach Abschluß der Funktion ermittelt werden, ob die Funktion ordnungsgemäß ausgeführt wurde. Wurde sie das nicht, erhält man in der Rückgabevariablen einen Minuswert.

| | |
|---|---|
| -32 | Funktions-Opcode ist ungültig |
| -33 | Datei wurde nicht gefunden |
| -34 | Pfadbezeichnung wurde nicht gefunden |
| -35 | Zuviele Dateien offen |
| -36 | Kein Zugriff möglich |
| -37 | Handle ungültig |
| -39 | Speicher nicht ausreichend |
| -40 | Adresse ungültig |
| -46 | Laufwerk falsch |
| -49 | Weitere Dateien nicht vorhanden |

### Set DRV

aktuelles Laufwerk bestimmen

**A%=Gemdos(&HE,Drive%)**

Es kann hiermit ein beliebiges Laufwerk zum aktuellen erklärt werden. Mit *Drive%* wird die Kennziffer des gewünschten Laufwerks übergeben. Nach Funktionsende kann aus *A%* die Kennziffer des zuletzt aktuellen Laufwerks ermittelt werden.

Laufwerk A=0; B=1; C=3; D=4 usw.

## Current Disk

aktuelles Laufwerk ermitteln

**A%=Gemdos(&H19)**

Es kann hiermit ermittelt werden, welches Laufwerk gerade aktuell ist. Nach Funktionsende kann aus *A%* die Kennziffer des aktuellen Laufwerks ermittelt werden.

Laufwerk A=0; B=1; C=3; D=4 usw.

## Set Disk Transfer Adress

Diskbuffer-Adresse

**Reserve Fre(0)-256**
**Void Gemdos(&H1A,L:Himem)**

Ermöglicht die Bestimmung eines 44 Byte großen Speicherbereichs zum aktuellen DT-Buffer für GEMDOS-Diskettenoperationen. In diesem Fall wurde die DTA auf *Himem* gelegt. Bei Verwendung von z.B. "SEARCH FIRST" werden die relevanten Daten nun in dem neuen Buffer abgelegt. Er kann bei Bedarf wie folgt gelesen werden:

```
A$=Space$(44)
Bmove Himem,Varptr(A$),44
Print A$
```

## Get DTA

Diskbuffer-Adresse ermitteln

**A%=Gemdos(&H2F)**
**For I=0 To 43**
**  A$=A$+Chr$(Peek(A%+I))**
**Next I**

Ermittelt die Startadresse (DTA=DiskTransferAdress) des 44 Byte umfassenden Disk-Transfer-Buffers. Hier werden bei GEMDOS-Diskettenoperationen vom System verschiedene Daten abgelegt (z.B. Name der zuletzt gewählten Datei). Nach Funktionsende ist die Adresse in *A%* und der Bufferinhalt in *A$* abgelegt.

## Malloc — geschützten Speicher reservieren

**A%=Gemdos(&H48,L:-1)**
(Speichergröße erfragen)

**B%=Gemdos(&H48,L:A%)**
(Speicher reservieren)

Es kann ein Speicherbereich reserviert werden, der auch gegen System-Zugriffe geschützt ist. Außerdem läßt sich die noch verfügbare Speichergröße ermitteln. Beim zweiten Aufruf enthält *A%* die gesamte verfügbare Speichergröße. Es kann auch eine andere Größe angegeben werden. Die gewünschte Größe muß auf jeden Fall kleiner sein, als die noch vorhandene. In *B%* erhält man nach Funktionsende die Startadresse des reservierten Bereichs (für *MFREE* merken!). Wechselwirkungen mit *RESERVE* beachten!

## Mfree — reservierten Speicher freigeben

**Void Gemdos(&H49,L:Adresse%)**

Der durch *MALLOC* reservierte Speicherbereich wird wieder dem System zur Verfügung gestellt. In *Adresse%* ist der Funktion die Startadresse des Bereichs zu übergeben, der freigegeben werden soll.

## Sfirst — Datei suchen

**N.ame$="Name der Datei (evtl. mit Pfad)"+Chr$(0)**
**A%=Gemdos(&H4E,L:Varptr(N.ame$),&H0)**

Es wird im Directory nach der Datei mit dem angegebenen Namen gesucht. In *A%* erhält man nach Funktionsende entweder eine Null (Funktion ausgeführt) oder einen Minuswert (Fehlercodes siehe oben). Alle relevanten Daten werden nach Funktionsende im aktuellen "DiskTransfer-Buffer" abgelegt (Dateiname, Größe etc.). Die Dateigröße wird mit *SIZE%=LPEEK(GEMDOS(&H2F)+26)* ermittelt.

# 14.5 BIOS

## GETMPB

MPB-Adresse ermitteln

**A$=Space$(12)**
**Void Bios(0,L:Varptr(A$))**
**Mfl%=Lpeek(Varptr(A$))**
**F.ree.s%=Lpeek(Mfl%+4)**
**F.ree.l%=Lpeek(Mfl%+8)**

Es kann die Adresse des "Memory-Parameter-Blocks" bestimmt und dieser dann analysiert werden. In *A$* wird ein 12 Byte großer Leerstring eingerichtet, der nach Funktionsende den MPB aufnimmt. Die Struktur des Blocks soll hier nicht näher erläutert werden. Aus dem MPB wird in diesem Fall anschließend in *F.ree.s%* die Startadresse des freien Speichers und in *F.ree.l%* dessen Größe ermittelt.

## RWABS

Disksektoren schreiben/lesen

**A$=Space$(Anz%*512)**
**Void Bios(4,Rwf%,L:Varptr(A$),Anz%,Rec%,Dev%)**

Es können Diskettensektoren gelesen oder beschrieben werden. Es wird mit *A$* ein Buffer eingerichtet, der die zu lesenden/schreibenden Daten erhält/enthält. Dieser Buffer muß 512 mal so groß sein, wie Sektoren gelesen/geschrieben werden sollen (1 Sektor=512 Byte). *Rwf%* bestimmt, ob gelesen (0) oder geschrieben (1) werden soll. *Anz%* enthält die Anzahl der Sektoren, die nacheinander gelesen/geschrieben werden sollen. *Rec%* bestimmt, mit welchem Sektor begonnen werden soll (1-720). *Dev%* enthält die Laufwerkskennziffer (0=A;1=B;2=Harddisk).

Sollen Sektoren gelesen werden, kann mann die Daten anschließend mit *PRINT A$* ausgeben lassen.

## GETBPB — BIOS-Parameterblock-Adresse ermitteln

```
A%=Bios(7,Dev%)
For I=0 To 7
  Print Dpeek(A%+I*2)
Next I
```

Es wird die Adresse des 18 Byte großen BIOS-Parameterblocks für das angegebene Laufwerk zurückgegeben. In *Dev%* wird die Kennziffer des gewünschten Laufwerks (0=A; 1=B) übergeben. *A%* enthält nach Abschluß der Funktion die BPB-Adresse. Die hier ausgegebenen Werte sind der Reihenfolge nach:

Bytegröße eines Sektors
Cluster pro Sektor
Bytegröße eines Clusters
Directory-Länge in Sektoren
Größe eines FAT in Sektoren
Sektornummer des 2. FAT
Sektornummer des 1. Daten-Clusters
Anzahl der vorhandenen Daten-Cluster

## MEDIACH — Diskettenwechsel feststellen

```
A%=Bios(9,Dev%)
```

Es kann festgestellt werden, ob die Diskette im aktuellen Laufwerk entfernt wurde oder nicht. *Dev%* enthält die Kennziffer des zu überprüfenden Laufwerks (0=A; 1=B). Nach Funktionsende können in A% folgende Werte auftreten:

| | |
|---|---|
| 0 | Diskette wurde nicht gewechselt |
| 1 | Diskette kann gewechselt worden sein |
| 2 | Diskette wurde gewechselt |

## DRVMAP — angeschlossene Laufwerke ermitteln

**A%=Bios(10)**

Es kann ermittelt werden, welche Laufwerke z.Zt. angeschlossen sind. Folgende Werte können in *A%* auftreten:

| | |
|---|---|
| 3 | A und B sind verfügbar |
| 7 | A, B, und C sind verfügbar |
| 11 | A, B, und D sind verfügbar |
| 15 | A, B, C und D sind verfügbar |

Auch wenn kein Laufwerk oder nur Laufwerk A angeschlossen ist, wird trotzdem der Wert 3 geliefert, da davon ausgegangen wird, daß immer mindestens A und B verfügbar sind.

## KBSHIFT — Switch-Tastenstatus ermitteln/setzen

**A%=Bios(11,Key%)**

Es kann der Status der Wechseltasten wahlweise ermittelt oder bestimmt werden. Ein positiver Wert in *Key%* wird als Tastenstatus übernommen (Werte siehe unten). Wird in *Key%* der Wert -1 übergeben, wird der aktuelle Status ermittelt. Tastenkombinationen werden durch Addition der Werte dargestellt. Folgende Werte können in *A%* auftreten:

| | |
|---|---|
| 1 | rechte Shift-Taste gedrückt |
| 2 | linke Shift-Taste gedrückt |
| 4 | Control-Taste gedrückt |
| 8 | Alternate-Taste gedrückt |
| 16 | CapsLock an |

## PHYSBASE — phys. Bildschirmadresse ermitteln

**A%=Xbios(2)**

Es wird die physikalische Basisadresse des Bildschirmspeichers zurückgegeben. Die Adresse steht nach Funktionsende in *A%*.

## LOGBASE — logische Bildschirmadresse ermitteln

**A%=Xbios(3)**

Es wird die logische Basisadresse des Bildschirmspeichers zurückgegeben. Die Adresse steht nach Funktionsende in *A%*.

## GETREZ — Bildschirmauflösung ermitteln

**A%=Xbios(4)**

Es wird die aktuelle Auflösung des Bildschirms ermittelt. In *A%* können folgende Werte auftreten:

| | | |
|---|---|---|
| 0 | = | 320 * 200 Punkte |
| 1 | = | 640 * 200 Punkte |
| 2 | = | 640 * 400 Punkte |

## SETSCREEN

Screen-Parameter setzen

**Void=Xbios(5,L:L.ad%,L:P.ad%,Res%)**

Es können die Auflösung, sowie logische und physikalische Bildschirmadresse verändert werden.

| | |
|---|---|
| L.ad% | enthält die neue *Logbase* |
| P.ad% | enthält die neue *Physbase* |
| Res% | enthält die neue Auflösung |

Soll einer der Werte unverändert bleiben, muß -1 übergeben werden.

## SETCOLOR

Farbregister lesen/setzen

**A%=Xbios(7,Reg%,Col%)**

Es können einzelne Farben gesetzt, bzw. gelesen werden. Die Variante *Farbe setzen* ist identisch mit dem BASIC-Befehl *SETCOLOR*. Es kann allerdings auch der aktuelle Farbwert des angegebenen Farbregisters ermittelt werden. *Reg%* enthält die gewählte Registernummer *Col%* ist der zu setzenden Farbwert (0-1911) Soll der aktuelle Farbwert ermittelt werden, ist in *Col%* der Wert -1 zu übergeben. Anschließend kann der Wert aus *A%* ausgelesen werden.

**A$=Space$(512*Anz%)**
**Adr%=Varptr(A$)**
**A%=Xbios(8,L:Adr%,L:1,Dev%,Sec%,Trk%,Sid%,Anz%)**

Es können ein oder mehrere Sektoren von Diskette gelesen werden. Mit *A$* wird ein Puffer vorbereitet, der die gelesenen Zeichen aufnehmen soll.

| | |
|---|---|
| Anz% | Anzahl der zu lesenden Sektoren (1-9) |
| Adr% | Adresse des Puffers (hier Varptr(A$)) |
| Dev% | Nummer der gewählten Station (0=A; 1=B) |
| Sec% | Nummer des ersten Sektors, der gelesen werden soll (0 - 8) |
| Trk% | Nummer des Tracks, der die zu lesenden Sektoren enthält (0 - 79/81) |
| Sid% | Diskettenseite, auf welcher der Track steht (0 = einseitig / 0/1 = zweiseitig |

Nach Funktionsende können die gelesenen Zeichen durch *PRINT A$* ausgegeben werden. In *A%* wird von der Funktion ein Code abgelegt.

Bedeutung:

| | |
|---|---|
| 0 | Kein Fehler aufgetreten |
| -1 | Allgemeiner Fehler |
| -2 | Station nicht empfangsbereit |
| -3 | Unbekannter Befehl |
| -5 | Ungültiger Befehl |
| -6 | Track nicht gefunden |
| -7 | Bootsektor nicht gültig |
| -8 | Sektor nicht gefunden |
| -10 | Schreibfehler |
| -12 | wie -1 |
| -13 | Floppy ist schreibgeschützt |
| -14 | Floppy wurde gewechselt |
| -15 | Gerät unbekannt |
| -16 | Fehlerhafter Sektor |
| -17 | Floppy nicht eingelegt |

## FLOPWR Sektor(en) schreiben

**A$="Enthält die zu schreibenden Zeichen"**
**Adr%=Varptr(A$)**
**A%=Xbios(9,L:Adr%,L:1,Dev%,Sec%,Trk%,Sid%,Anz%)**

Es können ein oder mehrere Sektoren auf Diskette geschrieben werden. Mit *A$* wird ein Puffer vorbereitet, der die zu schreibenden Zeichen enthält. Er muß 512 mal so groß sein, wie die Anzahl der zu beschreibenden Sektoren (*Anz%*). Die zu übergebenden Parameter haben dieselbe Bedeutung, wie bei *FLOPRD* beschrieben wurde. In *A%* wird ein Wert zurückgegeben, der ebenfalls dieselbe Bedeutung wie oben hat.

## FLOPFMT Track formatieren

**A$=Space$(8000)**
**Adr%=Varptr(A$)**
**A%=Xbios(10,L:Adr%,L:1,Dev%,Spt%,Trk%,Sid%,1,L:&H8 7654321,$HE5E5)**

Es kann ein einzelner Track formatiert werden. Mit *A$* wird ein Puffer vorbereitet, der die erzeugten Trackdaten aufnehmen soll (mind. 8000 Byte).

Spt% = Sektoren pro Track (üblich 9)

Alle weiteren zu übergebenden Variablen haben dieselbe Bedeutung, wie bei *FLOPRD* beschrieben wurde. Nach Funktionsende können die Trackdaten durch *PRINT A$* ausgegeben werden. In *A%* wird von der Funktion ein Code abgelegt, der ebenfalls dieselbe Bedeutung wie oben hat. Um eine Diskette komplett zu formatieren, muß ein Bootsektor erzeugt werden. Diesen hier zu erklären, würde zu weit führen. Mit dieser Funktion allein lassen sich nur einzelne Tracks im aktuellen Format formatieren.

## MIDIWS — String an MIDI übergeben

**A$="Zu sendender Midi-String"**
**Void Xbios(12,Len(A$)-1,L:Varptr(A$))**

Es wird ein String definiert, der dann durch diese Funktion an *MIDIOUT* gesandt wird.

## FLOPVER — Sektor(en) verifizieren

**A$="Enthält die zu vergleichenden Zeichen"**
**Adr%=Varptr(A$)**
**A%=Xbios(19,L:Adr%,L:1,Dev%,Sec%,Trk%,Sid%,Anz%)**

Es können ein oder mehrere Sektoren auf Diskette mit Pufferinhalt verglichen werden. Mit *A$* wird ein Puffer vorbereitet, der die zu vergleichenden Zeichen enthält. Er muß 512 mal so groß sein, wie die Anzahl der zu vergleichenden Sektoren (*Anz%*). Üblicherweise wird diese Funktion verwendet, um festzustellen, ob die mit *FLOPRD* oder *FLOPWR* gelesenen/geschriebenen Bytes exakt übertragen wurden. Die zu übergebenden Parameter haben dieselbe Bedeutung, wie bei *FLOPRD* beschrieben wurde. In *A%* wird ein Wert zurückgegeben, der ebenfalls dieselbe Bedeutung wie dort hat.

## SETPRT — Druckerattribute lesen/setzen

**A%=Xbios(33,Attr%)**

Es können verschiedene Druckereinstellungen gelesen oder gesetzt werden. Sollen die Attribute gesetzt werden, wird mit *Attr%* ein 16-Bit-Wert übergeben. Wird in *Attr%* der Wert -1 eingesetzt, wird von der Funktion in *A%* die aktuelle Einstellung als 16-Bit-Wert zurückgegeben. Die ersten 6 Bits haben folgende Bedeutung:

| | gesetzt | nicht gesetzt |
|---|---|---|
| Bit 1 | Matrixdrucker | Typenraddrucker |
| Bit 2 | Colordrucker | Schwarz/Weiß-Drucker |
| Bit 3 | Atari | Epson |
| Bit 4 | Test | Maximum |
| Bit 5 | Centronics | RS232 |
| Bit 6 | Endlos | Einzelblatt |

Die Bits 7-16 sind nicht interessant. Mit dieser Funktion können dieselben Einstellungen vorgenommen werden, wie sie vom Accessory "Druckeranpassung" bekannt sind.

## KBRATE — Tastatur-Repeat einstellen

**A%=Xbios(35,Vrz%,Rep%)**

Setzt Reaktionsverzögerung und Repeattakt für Tastaturklick. *Vrz%* enthält den Verzögerungswert. *Rep%* enthält den Wiederholungswert Der Wert 1 steht für sehr schnell. Je größer der Wert, umso langsamer. Wird in *Rep%* der Wert 0 übergeben, sind nur einzelne Tastenklicks möglich. Soll ein Parameter unverändert bleiben, ist dafür der Wert -1 zu verwenden. Nach Funktionsende wird von der Funktion in *A%* ein Zweibyte-Wert zurückgegeben. Im Highbyte steht dann der vorherige Repeat und im Lowbyte der vorherige Verzögerungswert.

## PUNTAES — AES abschalten

**Void Xbios(39)**

Befindet sich das AES im RAM, wird es hiermit abgeschaltet.

# 15. Anhang

## 15.1 Direktmodus

Grundsätzlich ist es möglich, jeden GfA-BASIC-Befehl im Direktmodus (Direktfeld mit der Maus anklicken oder die <ESC>-Taste drücken) einzugeben.

Ausnahme: Schleifen- und GOTO-Anweisungen, PROCEDURE-Definitionen und Bedingungsabfragen.

Die Rückkehr aus dem Direktmodus zum Editor erfolgt wahlweise mit <ESC> oder Eingabe von *ED*, gefolgt von <RETURN> oder durch gleichzeitiges Drücken der Tasten <CONTROL> <SHIFT> und <ALTERNATE>.

GfA-BASIC bietet die Möglichkeit, eigene Editor-Routinen zu schreiben und diese aus dem Direktmodus aufzurufen, da der Befehl *GOSUB* hier auch funktioniert und somit eine Prozedur, die sich im aktuellen Programm befindet, aufgerufen werden kann. So kann unter Umständen ein Verlassen des Interpreters vermieden werden, da das GfA-BASIC Befehle zur Verfügung stellt, welche interpreter-externe Disketten- oder Druckeroperationen erlauben.

Da beim Programmieren mit GfA-BASIC oft Endlosschleifen benutzt werden, muß erwähnt werden, daß ein Programmende grundsätzlich (Ausnahme siehe bei *ON BREAK*) erreicht wird, wenn die Tasten <CONTROL>, <SHIFT> und <ALTERNATE> gleichzeitig gedrückt werden.

## 15.2 Der Editor

### Cursor-Positionierung

**Cursor rechts**
Editorcursor in der aktuellen Programmzeile um eine Position nach rechts bewegen.

Pfeil-rechts-Taste im Cursor-Tastenblock drücken.

**Cursor links**
Editorcursor in der aktuellen Programmzeile um eine Position nach links bewegen .

Pfeil-links-Taste im Cursor-Tastenblock drücken.

**Cursor an Zeilenanfang**
Editorcursor in der aktuellen Programmzeile zum Zeilenanfang bewegen.

<CONTROL>-Taste und Pfeil-links-Taste gleichzeitig drücken.

**Cursor an Zeilenende**
Editorcursor in der aktuellen Programmzeile zum Zeilenende bewegen.

<CONTROL>-Taste und Pfeil-rechts-Taste gleichzeitig drücken.

**Cursor-rechts-Tabulator**
Editorcursor in der aktuellen Programmzeile in 8er-Schritten nach rechts bewegen.

<TAB>-Taste drücken.

**Cursor-Links-Tabulator**
Editorcursor in der aktuellen Programmzeile in 8er-Schritten nach links bewegen.

<CONTROL>-Taste und <TAB>-Taste gleichzeitig drücken.

**Cursor eine Zeile aufwärts**
Editorcursor um eine Zeile über die aktuelle Programmzeile bewegen und Syntaxkontrolle.

Pfeil-oben-Taste im Cursor-Tastenblock drücken.

**Cursor eine Zeile abwärts**
Editorcursor um eine Zeile unter die aktuelle Programmzeile bewegen und Syntaxkontrolle.

Pfeil-unten-Taste oder <RETURN>-Taste drücken.

**Cursor beliebig prositionieren**
Editorcursor auf der aktuellen Seite an eine beliebige Stelle bewegen und Syntaxkontrolle bei Zeilenwechsel.

Anklicken der gewünschten Position mit dem Mauszeiger.

**Cursor an Seitenanfang**
Editorcursor an den Anfang der aktuellen Editorseite bewegen, Syntaxkontrolle und Strukturierung.

<CLRHOME>-Taste im Cursor-Tastenblock drücken.

**Cursor eine Seite rückwärts**
Editorcursor an den Anfang der vorhergehenden Editorseite bewegen, Syntaxkontrolle und Strukturierung.

<CONTROL>-Taste und Pfeil-oben-Taste gleichzeitig oder <SHIFT>-Taste und <F7>-Taste gleichzeitig drücken oder *PG UP* im Editormenü anklicken.

**Cursor eine Seite vorwärts**
Editorcursor an den Anfang der nächsten Editorseite bewegen, Syntaxkontrolle und Strukturierung.

<CONTROL>-Taste und Pfeil-unten-Taste gleichzeitig oder <F7>-Taste drücken oder *PG DOWN* im Editormenü anklicken.

**Cursor an Programmanfang**
Editorcursor an den Anfang des Programms bewegen, Syntaxkontrolle und Strukturierung.

<CONTROL>-Taste und <CLRHOME>-Taste gleichzeitig drücken.

**Cursor an Programmende**
Editorcursor an das Ende des Programms bewegen, Syntaxkontrolle und Strukturierung.

<CONTROL>-Taste und <Z>-Taste gleichzeitig drücken.

## Das Editormenü

**Programm speichern**
Programm wird mit der Extension *".BAS"* auf Diskette gespeichert. Dabei wird das Programm als gesamter Speicherblock direkt auf die Diskette verfrachtet, ohne die Interpreter-Kontrolle zu durchlaufen.Dadurch ergibt sich eine erheblich kürzere Speicherzeit.

<SHIFT>-Taste und <F1>-Taste gleichzeitig drücken oder *SAVE* im Editormenü anklicken.

**Programm laden**
Programme mit der Extension *".BAS"* können von Diskette geladen werden. Hier wird ebenfalls die Interpreterkontrolle übersprungen. Die hiermit geladenen Dateien verdrängen die im Editor befindlichen Programme. Gegebenenfalls sollten Sie das aktuelle Programm im Editor zuerst auf Diskette sichern. Der LOAD-Vorgang läßt sich nach der Dateiauswahl in der Fileselectbox nur noch durch einen Reset stoppen. Dadurch wäre das Editorprogramm allerdings ebenfalls unwiderruflich verloren.

<F1>-Taste drücken oder *LOAD* im Editormenü anklicken.

**Textdatei speichern**
Programm wird als Textfile mit der Extension *".LST"* auf Diskette gespeichert. Dabei durchläuft die geladene Datei die Interpreter-Kontrolle.Die benötigte Zeit für den Speichervorgang ist erheblich länger, als mit *SAVE*.

<SHIFT>-Taste und <F2>-Taste gleichzeitig drücken oder *SAVE,A* im Editormenü anklicken.

**Textdatei laden**
Als Textdatei abgespeicherte Programme mit der Extension *".LST"* können von Diskette geladen werden. Auch diese Funktion benötigt mehr Zeit zum Laden, als die LOAD-Funktion. Sie hat dagegen allerdings den Vorteil, daß verschiedene Dateien miteinander verknüpft werden können.

<F2>-Taste drücken oder *MERGE* im Editormenü anklikken.

**Interpreter beenden**
Es erscheint eine Kontrollabfrage. Antwort *JA* bewirkt Rückkehr zum Desktop. Antwort *NEIN* kehrt zum Interpreter zurück.

<SHIFT>-Taste und <F3>-Taste gleichzeitig drücken oder *QUIT* im Editormenü anklicken.

**Programmlisting ausdrucken**
Es erscheint eine Kontrollabfrage. Antwort *JA* druckt formatiertes Listing auf angeschlossenem Drucker aus. (Vorsicht: sollte kein Drucker angeschlossen sein oder ist er nicht "On Line" kehrt das System erst nach ca. einer halben Minute zum Interpreter zurück. Also, nicht gleich den RESET-Knopf drücken!) Antwort *NEIN* kehrt zum Interpreter zurück.

<F3>-Taste drücken oder *LLIST* im Editormenü anklicken.

**Programm löschen**
Es erscheint eine Kontrollabfrage. Antwort *JA* löscht das im Editor befindliche Programm unwiderruflich. Antwort *NEIN* kehrt zum Editor zurück.

<SHIFT>-Taste und <F4>-Taste gleichzeitig drücken oder *NEW* im Editormenü anklicken.

**Blockoperationen**

Wurde mit *BLK STA* und *BLK END* ein Textblock definiert, so erscheint das Blockmenü. Fehlt eine der beiden Definitionen so erscheint "Block ???".

<F4>-Taste drücken oder *BLOCK* im Editormenü anklikken.

**Block kopieren**
Ein Duplikat des definierten Blockes wird an die Stelle unmittelbar über der aktuellen Cursorposition übertragen. Der Originalblock bleibt unverändert an seiner Position.

<C>-Taste drücken oder *COPY* im Blockmenü anklicken.

**Block bewegen**
Der definierte Block wird an die Stelle unmittelbar über der aktuellen Cursorposition bewegt. Dabei wird er an der Ursprungsposition gelöscht. Die Blockdefinition wird gleichzeitig aufgehoben.

<M>-Taste drücken oder *MOVE* im Blockmenü anklicken.

**Block speichern**
Der definierte Block wird (wieder mittels Fileselectbox) als ".LST"-Datei auf Diskette abgespeichert. Der Block bleibt im Editor unverändert erhalten.

<W>-Taste drücken oder *WRITE* im Blockmenü anklicken.

**Block ausdrucken**
Der definierte Block wird ausgedruckt. Hinweis zu "Programmlisting ausdrucken" gilt hier analog.

<L>-Taste drücken oder *LLIST* im Blockmenü anklicken.

**Blockanfang finden**
Der Cursor wird am Blockanfang positioniert.

<S>-Taste drücken oder *START* im Blockmenü anklicken.

**Blockende finden**
Der Cursor wird unmittelbar unter dem Blockende positioniert

<E>-Taste drücken oder *END* im Blockmenü anklicken.

**Block löschen**
Der definierte Block wird ohne Kontrollabfrage sofort und unwiderruflich aus dem Editor gelöscht. (Also: Vorsicht!)

<CONTROL>-Taste und <D>-Taste gleichzeitig drücken oder *DELETE* im Blockmenü anklicken.

**Blockanfang definieren**
Die Zeile, in welcher sich der Cursor befindet, wird zum Blockanfang erklärt.

<SHIFT>-Taste und <F5>-Taste gleichzeitig drücken oder *BLK STA* im Editormenü anklicken.

**Blockende definieren**
Die Zeile unmittelbar über dem Cursor wird zum Blockende erklärt.

<F5>-Taste drücken oder *BLK END* Im Editormenü anklicken.

**Ausdruck finden und ersetzen**
Es wird die Eingabe eines Zeichens oder einer Zeichenfolge erwartet, das/die im Programmtext unterhalb des Cursors gesucht werden soll. Anschliessend wird die Eingabe eines weiteren Zeichens oder einer Zeichenfolge erwartet, das/die den gesuchten Ausdruck ersetzen soll.

<SHIFT>-Taste und <F6>-Taste gleichzeitig drücken oder *REPLACE* im Editormenü anklicken. Im Anschluß an die Eingabe kann dann durch Drücken der Tastenkombination <CONTROL> und <F> die Suchfunktion aufgerufen werden. Soll der gefundene Ausdruck mit dem Ersatzausdruck ausgetauscht werden, drücken Sie die Tastenkombination <CONTROL> und <R>.

**Ausdruck finden**
Es wird die Eingabe eines zu suchenden Zeichens oder einer zu suchenden Zeichenfolge erwartet. Die Erläuterungen zu "Ausdruck finden und ersetzen" gelten hier analog. Gesucht wird durch gleichzeitiges Drücken von <CONTROL> und <F>.

<F6>-Taste drücken oder *FIND* im Editormenü anklicken.

**Seite rückwärts**
<SHIFT>-Taste und <F7>-Taste gleichzeitig drücken oder *PG UP* im Editormenü anklicken. (siehe auch bei Cursor-Funktionen)

**Seite rückwärts**
<F7>-Taste drücken oder *PG DOWN* im Editormenü anklicken. (siehe auch bei Cursor-Funktionen)

**Zeilenhöhe ändern**
Der Editortext kann entweder mit einer Schrifthöhe von 8 Pixels (PICture ELementS = engl. Bildelemente) in max. 48 Zeilen und 79 Spalten oder mit einer Schrifthöhe von 16 Pixels in max. 23 Zeilen und 79 Spalten dargestellt werden.

<SHIFT>-Taste und <F8>-Taste gleichzeitig drücken oder *TEXT 16* bzw. *TEXT 8* im Editormenü anklicken

**Texteingabemodus ändern**
Die eingegebenen Zeichen können verschiedene Auswirkungen auf den schon bestehenden Programmtext haben. Im *INSERT*-Modus (insert = engl. einfügen) wird das einzugebende Zeichen an der Cursorposition in den Text eingefügt. Der rechts davon liegende Zeilenrest wird um eine Stelle nach rechts verschoben. Im *OVERWRITE*-Modus (overwrite = engl. überschreiben) wird das Zeichen unter dem Cursor durch das neue Zeichen überschrieben.

<F8>-Taste drücken oder *INSERT* bzw. *OVERWRT* im Editormenü anklicken.

**Direktmodus einschalten**
Es wird in den sogenannten DIRECT- oder auch COMMAND-Modus umgeschaltet.

<SHIFT>-Taste und <F9>-Taste gleichzeitig drücken oder <ESC>-Taste drücken oder *DIRECT* im Editormenü anklicken.

**Ausgabefenster zeigen**
Das zweite Interpreterfenster, das zur Ausgabe von Grafik- oder Textoperationen dient, wird gezeigt. Dieser Vorgang bleibt ohne weitere Auswirkung auf die Editorarbeit. Nach Drücken einer beliebigen Taste oder einem Mausclick unterhalb des Editormenüs kehrt der Interpreter zum Editor zurück.

<F9>-Taste drücken oder *FLIP* im Editormenü anklicken.

**Programm starten**
Das im Editor befindliche Programm wird nach Strukturkontrolle gestartet.

<SHIFT>-Taste und <F10>-Taste gleichzeitig drücken oder *Run* im Editormenü anklicken.

**Schleifenstruktur testen**
Der Interpreter prüft, ob alle Schleifen ordnungsgemäß abgeschlossen wurden und kehrt zum Editor zurück.

<F10>-Taste drücken oder *TEST* im Editormenü anklicken.

**Weitere Hilfsfunktionen**

**Letzte Änderung löschen und Programmstruktur ordnen**
Wurde nach Änderung einer Programmzeile diese noch nicht mit <RETURN> oder einer Cursor-Vertikal-Pfeiltaste verlassen, kann diese Änderung wieder ungültig gemacht werden. Gleichzeitig wird das Programm komplett strukturiert. Wurden Programmteile verändert, ersetzt oder gelöscht, kann es sein, daß die sichtbare Programmstruktur nicht mehr korrekt geordnet ist. Mit dieser Funktion wird das Programm wieder in Ordnung gebracht. Beachten Sie jedoch die Anmerkung zur TEST-Funktion.

<UNDO>-Taste drücken

**Leerzeile einfügen**
Die aktuelle Cursorzeile wird um eine Position nach unten bewegt. Dadurch entsteht unter dem Cursor eine Leerzeile. Diese Zeile wird dann wie jede andere neue Zeile behandelt. Nach Texteingabe und <RETURN> als Zeilenabschluß wird solange eine neue Leerzeile eingefügt bis diese dann leer mit <RETURN> ,einer der Cursor-Vertikal-Pfeiltasten oder <UNDO> geschlossen wird.

<INSERT>-Taste drücken.

**Aktuelle Cursorzeile löschen**
Es wird die Zeile, in welcher sich der Cursor befindet, komplett und unwiderruflich gelöscht.

<CONTROL>-Taste und <DELETE>-Taste gleichzeitig drücken.

**Zeichen links vom Cursor löschen**
<BACKSPACE>-Taste drücken.

**Zeichen unter dem Cursor löschen**
<DELETE>-Taste drücken.

## 15.3 GfA-BASIC - Fehlerliste

**Editor-Fehlermeldungen**

WHILE OHNE WEND
REPEAT OHNE UNTIL
DO OHNE LOOP
FOR OHNE NEXT
WEND OHNE WHILE
LOOP OHNE DO
NEXT OHNE FOR
IF OHNE ENDIF
ENDIF OHNE IF
ELSE OHNE IF
ELSE OHNE ENDIF
EXIT OHNE SCHLEIFE
PROCEDURE OHNE RETURN
PROCEDURE IN SCHLEIFE
PROCEDURE DOPPELT DEFINIERT
RETURN OHNE PROCEDURE
MARKE DOPPELT DEFINIERT
LOCAL NUR IN PROCEDURE
LOCAL NICHT IN SCHLEIFE
FUNKTION DOPPELT DEFINIERT
GOTO IN/AUS FOR-NEXT
GOTO IN/AUS PROCEDURE
RESUME IN FOR-NEXT-SCHLEIFE
RESUME OHNE PROCEDURE
SYNTAX FEHLER
ZEILE ZU LANG

| ERROR | TEXT |
|---|---|
| 0 | DIVISION DURCH NULL |
| 1 | ÜBERLAUF |
| 2 | ZAHL NICHT INTEGER -2147483648...2147483648 |
| 3 | ZAHL NICHT BYTE 0...255 |
| 4 | ZAHL NICHT WORT 0...65535 |
| 5 | QUADRATWURZEL NUR FÜR POSITIVE ZAHLEN |
| 6 | LOGARITHMEN NUR FÜR ZAHLEN > 0 |
| 7 | UNBEKANNTER FEHLER |
| 8 | SPEICHER VOLL |
| 9 | FUNKTION ODER BEFEHL NOCH NICHT MÖGLICH |
| 10 | STRING ZU LANG. MAX. 32767 ZEICHEN |
| 11 | KEIN GFA-BASIC V2.0 PROGRAMM |
| 12 | PROGRAMM ZU LANG, SPEICHER VOLL. NEW |
| 13 | KEIN GFA-BASIC. PROGRAMMFILE ZU KURZ. NEW |
| 14 | FELD ZWEIMAL DIMENSIONIERT |
| 15 | FELD NICHT DIMENSIONIERT |
| 16 | FELD INDEX ZU GROß |
| 17 | DIM INDEX ZU GROß |
| 18 | FALSCHE ANZAHL INDIZES |
| 19 | PROCEDURE NICHT GEFUNDEN |
| 20 | LABEL NICHT GEFUNDEN |
| 21 | BEI OPEN NUR ERLAUBT: "I"NPUT, "O"UTPUT, "R"ANDOM, "A"PPEND, "U"PDATE |
| 22 | FILE SCHON GEÖFFNET |
| 23 | FILE # FALSCH |
| 24 | FILE NICHT GEÖFFNET |
| 25 | FALSCHE EINGABE. KEINE ZAHL |
| 26 | FILEENDE ERREICHT EOF |
| 27 | ZUVIEL PUNKT FÜR POLYLINE/POLYFILL. MAX.128 |
| 28 | FELD MUß EINDIMENSIONAL SEIN |
| 29 | ANZAHL PUNKTE GRÖßER ALS FELD |
| 30 | MERGE - KEIN ASCII-FILE |
| 31 | MERGE - ZEILE ZU LANG - ABBRUCH |
| 32 | ==> SYNTAX NICHT KORREKT |
| 33 | MARKE NICHT DEFINIERT |
| 34 | ZUWENIG DATA |
| 35 | DATA NICHT NUMERISCH |
| 36 | SYNTAXFEHLER IN DATA. "" PAARWEISE VERWENDEN |

| | |
|---|---|
| 37 | DISKETTE VOLL |
| 38 | BEFEHL IM DIREKTMODUS NICHT MÖGLICH |
| 39 | PROGRAMMFEHLER. KEIN GOSUB MÖGLICH |
| 40 | CLEAR NICHT MÖGLICH IN FOR-NEXT-SCHLEIFEN ODER PROZEDUREN |
| 41 | CONT NICHT MÖGLICH |
| 42 | ZUWENIG PARAMETER |
| 43 | AUSDRUCK ZU KOMPLEX |
| 44 | FUNKTION NICHT DEFINIERT |
| 45 | ZUVIELE PARAMETER |
| 46 | PARAMETER FALSCH. MUß ZAHL SEIN |
| 47 | PARAMETER FALSCH. MUß STRING SEIN |
| 48 | OPEN "R" SATZLÄNGE FALSCH |
| 50 | KEIN "R"-FILE |
| 51 | NUR EIN FIELD ZU EINEM OPEN "R" MÖGLICH |
| 52 | FIELDS GRÖßER ALS SATZLÄNGE |
| 53 | ZU VIELE FIELDS (MAX. 9) |
| 54 | GET/PUT FIELD-STRINGSLÄNGE FALSCH |
| 55 | GET/PUT SATZNUMMER FALSCH |
| 60 | SPRITE-STRINGLÄNGE FALSCH |
| 61 | FEHLER BEI RESERVE |
| 62 | FEHLERHAFTES MENÜ |
| 63 | FEHLERHAFTES RESERVE |
| 64 | FEHLERHAFTER POINTER |
| 90 | FEHLER BEI LOCAL |
| 91 | FEHLER BEI FOR |
| 92 | RESUME (NEXT) NICHT UNMÖGLICH. FATAL, FOR ODER LOCAL |
| 100 | C COPYRIGHT 1986, GFA SYSTEMTECHNIK |

# TOS-Fehlermeldungen

| ERROR | TEXT |
|---|---|
| -1 | ALLGEMEINER FEHLER |
| -2 | DRIVE NOT READY. ZEITÜBERSCHREITUNG |
| -3 | UNBEKANNTER BEFEHL |
| -4 | CRC FEHLER, DISK-PRÜFSUMME FALSCH |
| -5 | BAD REQUEST. UNGÜLTIGER BEFEHL |
| -6 | SEEK ERROR. SPUR NICHT GEFUNDEN |
| -7 | UNKNOWN MEDIA. FALSCHER BOOTSEKTOR |
| -8 | SEKTOR NICHT GEFUNDEN |
| -9 | KEIN PAPIER |
| -10 | SCHREIBFEHLER |
| -11 | LESEFEHLER |
| -12 | ALLGEMEINER FEHLER 12 |
| -13 | DISKETTE SCHREIBGESCHÜTZT |
| -14 | DISKETTE WURDE GEWECHSELT |
| -15 | UNBEKANNTES GERÄT |
| -16 | BAD SEKTOR (VERIFY) |
| -17 | ANDERE DISKETTE EINLEGEN |
| -32 | UNGÜLTIGE FUNKTIONSNUMMER |
| -33 | DATEI NICHT GEFUNDEN |
| -34 | PFADNAME NICHT GEFUNDEN |
| -35 | ZUVIELE DATEIEN OFFEN |
| -36 | ZUGRIFF NICHT MÖGLICH |
| -37 | UNGÜLTIGES HANDLE |
| -39 | SPEICHER VOLL |
| -40 | UNGÜLTIGE SPEEICHERBLOCKADRESSE |
| -46 | UNGÜLTIGE LAUFWERKSBEZEICHNUNG |
| -49 | KEINE WEITEREN DATEN |
| -64 | GEMDOS BEREICHSFEHLER. SEEK FALSCH? |
| -65 | INTERNER GEMDOS-FEHLER |
| -66 | KEIN BINÄRPROGRAMMFILE |
| -67 | SPEICHERBLOCKFEHLER |

## Bomben-Fehlermeldungen

| ERROR | TEXT |
|---|---|
| 102 | 2 BOMBEN - BUS ERROR. VIELLEICHT PEEK ODER POKE FALSCH |
| 103 | 3 BOMBEN - ADRESS ERROR. UNGERADE WORTADRESSE! BEI DPOKE, DPEEK, LPOKE ODER LPEEK |
| 104 | 4 BOMBEN - ILLEGAL INSTRUCTION. AUSFÜHRUNG EINES UNGÜLTIGEN 68000-MASCHINENBEFEHLS |
| 105 | 5 BOMBEN - DIVIDE BY CERO. DIVISION DURCH NULL IN 68000-MASCHINENSPRACHE |
| 106 | 6 BOMBEN - CHK-EXEPTION. 68000-INTERRUPT DURCH CHK-BEFEHL |
| 107 | 7 BOMBEN - TRAPV-EXEPTION. 68000-INTERRUPT DURCH TRAPV-BEFEHL |
| 108 | 8 BOMBEN - PRIVILEGE VIOLATION. 68000-INTERRUPT DURCH AUSFÜHRUNG EINES PRIVILIGIERTEN BEFEHLS |
| 109 | 9 BOMBEN - TRACE-EXEPTION. 68000-TRACE-INTERRUPT |

## 15.4 Quick-Reference


| Befehlsname, Syntax | Abk. | Seite |
|---|---|---|
| ! Kommentar | | 165 |
| *var | | 166 |
| ABS(x) | | 87 |
| ADD V_name1,V_name2<br>ADD V_name1,const | [Ad] | 85 |
| ADDRIN | | 154 |
| ADDROUT | | 154 |
| ALERT Icon%,B_oxtext$,Def_Button%,Buttontext$,Backvar% | [A] | 134 |
| ARRAYFILL Array(),x | [Ar] | 123 |
| ARRPTR(Stringname$)<br>APRPTR(Feldname()) | | 133 |
| ASC("Zeichen"/Stringvariable) | | 116 |
| ATN(x) | | 88 |
| BASEPAGE | | 166 |
| BGET #Kanal,Adresse,Anzahl | [Bg] | 168 |
| BIN$(Ausdruck)<br>&XBinärausdruck | | 117 |
| BIOS(Opcode,Parameterliste) | | 153 |
| BITBLT Q.db%(),Z.db%(),R.db%() | [Bit] | 167 |
| BLOAD "Dateiname",Start | [Bl] | 32 |
| BMOVE Quelladresse,Zieladresse,Anzahl | [Bm] | 169 |
| BOX X_links,Y_oben,X_rechts,Y_unten | [B] | 105 |
| BPUT #Kanal,Adresse,Anzahl | [Bp] | 169 |
| BSAVE "Dateiname",Start,Anzahl | [Bs] | 32 |

C: ..... 127

CALL ..... [Ca] 128

CHAIN "Programmname" ..... [Ch] 33

CHDIR "Ordnername" ..... [Chd] 33

CHDRIVE Laufwerk ..... [Chdr] 34

CHR$(Wert) ..... 117

CIRCLE X_pos,Y_pos,Radius ..... [C] 106
CIRCLE X_pos,Y_pos,Radius,Alpha,Beta

CLEAR ..... [Cle] 64

CLEARW Handle ..... [Cle W] 137

CLOSE #Kanalnummer ..... [Cl] 45

CLOSEW Handle ..... [Cl W] 137

CLR Var,Var%,Var$ ..... 169

CLS #Kanalnummer ..... 64

COLOR Farbe ..... [Co] 95

CONT ..... [Con] 65

CONTRL ..... 154

COS(x) ..... 88

CRSCOL ..... 170

CRSLIN ..... 170

CVI(2-Zeichen-Stringausdruck) ..... 118

CVL(4-Zeichen-Stringausdruck) ..... 118

CVS(4-Zeichen-Stringausdruck) ..... 118

CVF(6-Zeichen-Stringausdruck) ..... 118

CVD(8-Zeichen-Stringausdruck) ..... 118

DATA Eintrag1,"Texteintrag1" ..... [D] 65

DATE$ ..... 155

DEC V_name ..... 85

DEFFILL Farbe,Füllstil,Muster ..... [Deff] 96
DEFFILL Farbe,F_muster$

DEFFN Funktionsname=Funktionsausdruck ..... 66
DEFFN Funktionsname(Variablenliste)=Funktionsausdruck

DEFLINE Linienstil,Liniendicke,Anfangsform,Endform ..... [De] 98

DEFLIST ..... [Deflis] 67

DEFMARK Markerfarbe,Markertyp,Markergröße ..... [Defm] 99

DEFMOUSE Form ..... [Defmo] 99
DEFMOUSE Maus$

DEFNUM Stelle ..... [Defn] 170

DEFTEXT Textfarbe,Textart,Textwinkel,Textgröße ..... [Deft] 101

DFREE(Laufwerk) ..... 32

DIM V_name(index1,index2,..) ..... [Di] 123
DIM V_name1(indizes),V_name2(indizes)

DIM?(Array()) ..... 124

DIR "Dateienbezeichnung" TO "Dateiname"/"LST:" ..... 34

DIR$(Laufwerk) ..... 35

DIV V_name1,V_name2 ..... 85
DIV V_name1,const

DO LOOP ..... [Do..L] 56

DPEEK(gerade Adresse) ..... 125

DPOKE gerade Adresse,Word ..... 126

DRAW (TO) X_pos,Y_pos ..... [Dr] 107
DRAW X1,Y1 TO X2,Y2 TO Xn,Yn

EDIT ..... [Ed] 67

ELLIPSE X_pos,Y_pos,A,B ..... [Ell] 107
ELLIPSE X_pos,Y_pos,A,B,Alpha,Beta

END ..... 68

EOF (#Kanalnummer) ..... 40

ERASE Array() ..... [Er] 124

ERR ..... 156

ERROR Fehlernummer ..... [Err] 156

EVEN(Zahl) ..... 171

EXEC Modus,"Prg","Kom","Env" ..... 171

EXIST(Dateiname) ..... 36

EXIT IF Bedingung ..... [E] 59

EXP(x) ..... 88

FALSE ..... 173

FATAL ..... 156

FIELD / AS ..... [Fi / As] 45

FILES "Dateienbezeichnung" TO "Dateiname"/"LST:" ..... [File] 36

FILESELECT "Pfadname","Vorauswahlangabe",Backvar$ ... [Filese] 135

FILL X_pos,Y_pos ..... [Fi] 108

FIX(x) ..... 89

FN Funktionsname(Parameterliste) ..... [@] 68

FOR Zähler = Anfang TO/DOWNTO Ende STEP Schritt ..... [F...N] 56
NEXT Zähler

FORM INPUT Anzahl,V_name$ ..... [F minput] 21

FORM INPUT Anzahl AS Var$ ..... [F minput as] 173

FRAC(x) ..... 89

FRE(Dummy) ..... 157

FULLW Handle ..... [Fu] 137

GB ..... 157

GCONTRL ..... 158

GEMDOS(Opcode,Parameterliste) ..... 153

GEMSYS Opcode ..... 151

GET X1,Y1,X2,Y2,V_name$ ..... 114

GET #Kanalnummer,Satznummer ..... 46

GINTIN ..... 158

GINTOUT ..... 159

GOSUB Procedurename(Parameterliste) ..... [Go bzw. @] 59

GOTO Label ..... [Got] 60

GRAPHMODE Modus ..... [G] 103

HARDCOPY ..... [H] 43

HEX$(Ausdruck) ..... 119
&Hhexadezimalausdruck

HIDEM ..... [Hi] 173

HIMEM ..... 174

IF..ELSE..ENDIF ..... [I..El..En] 60

INC V_name ..... [In] 85

INFOW Handle,Stringausdruck ..... [Inf] 138

INKEY$ ..... 21

INP (Port) ..... 46
INP (#Kanalnummer)

INP?(Port) ..... 174

INPUT "Text";V_name1,V_name2 ..... [Inp] 23

INPUT #Kanalnummer,V_name ..... 47

INPUT$(Anzahl,#Kanal) ..... 24

INSTR(Startposition,Zielstring,Suchstring ..... 76

INT(x) ..... 89

INTIN ..... 159

INTOUT ..... 159

KILL "Dateiname" ..... [K] 37

LEFT$(Zielstring,Anzahl) ..... 76

LEN(V_name$) ..... 77

LET V_name=Ausdruck ..... [Le] 69
LET V_name$=Textausdruck

LINE X1,Y1,X2,Y2 ..... [Li] 108

LINE INPUT "Text"[;,]V_name$ ..... [Li input] 25

LINE INPUT #Kanalnummer,V_name$ ..... 48

LIST "Dateiname" ..... [Lis] 37

LLIST ..... [Ll] 43

LOAD "Programmname" ..... [Loa] 37

LOCAL Lv,Lv%,Lv$ ..... [Loc] 69

LOC (#Kanalnummer) ..... 40

LOF (#Kanalnummer) ..... 41

LOG(x) / LOG10(x) ..... 90

LPEEK(gerade Adresse) ..... 125

LPOKE gerade Adresse,Long ..... [Lp] 126

LPOS(Dummy) ..... 44

LPRINT "Text"[;,']V_name[;,']V_name$[;,'] ..... [Lpr] 44

LSET Ziel_var$="Stringausdruck" ..... [Ls] 77

MAX(Ausdruck1,Ausdruck2,..) ..... 50

MENU(Index) ..... 144

MENU Menü,Wert ..... 146

MENU Stringfeld() ..... [Me array$()] 147

MENU KILL ..... [Me Kill] 148

MENU OFF ..... [Me Off] 148

MID$(Var$,Start,Anzahl)=Stringausdruck ..... 175

MID$(Zielstring,Start,Anzahl) ..... 78

MIN(Ausdruck1,Ausdruck2,..) ..... 91

MKDIR "Ordnername" ..... [Mk] 38

MKI$(16-Bit-Integer-Zahl) ..... 119

MKL$(32-Bit-Integer-Zahl) ..... 119

MKS$(Realzahl) ..... 119

MKF$(Realzahl) ..... 119

MKD$(Realzahl) ..... 119

MONITOR(Parameter) ..... [Mon] 175

MOUSE Xpos,Ypos,Tasten ..... [M] 136

MOUSEK ..... 136

MOUSEX ..... 136

MOUSEY ..... 136

MUL V_name1,V_name2 ..... [Mu] 86
MUL V_name1,const

NAME "Name_alt" AS "Name_neu" ..... [Na..As] 38

NEW ..... 70

OCT$(Ausdruck) ..... 120
&Ooktalwert

ODD(Zahl) ..... 176

ON Wert GOSUB Procedure1,Procedure2 ..... 62

ON BREAK GOSUB Procedurename ..... 61
ON BREAK CONT

ON ERROR GOSUB Procedurename ..... 61
ON ERROR

ON MENU ..... 148

ON MENU BUTTON Anzahl,Taste,Status GOSUB ..... 176

ON MENU GOSUB Procedurename ..... 149

ON MENU IBOX Id,Xp,Yp,Br,Hö GOSUB Procedurename ..... 149

ON MENU KEY GOSUB Procedurename ..... 150

ON MENU MESSAGE GOSUB Procedurename ..... 150

ON MENU OBOX Id,Xp,Yp,Br,Hö GOSUB Procedurename ..... 149

OPEN "Modus",#Kanalnummer,"Dateiname" ..... [O] 48

OPENW Handle,X_pos,Y_pos ..... [O W] 138
OPENW Handle

OPTION "Befehlstext" ..... [Opt] 178

OPTION BASE 0/1 ..... [Opt Base] 178

OUT Port,Bytewert ..... [Ou] 50
OUT #Kanalnummer,Bytewert

OUT?(Port) ..... 179

PAUSE Dauer ..... [Pa] 70

PBOX X_links,Y_oben,X_rechts,Y_unten ..... [Pb] 105

PCIRCLE X_pos,Y_pos,Radius ..... [Pc] 106
PCIRCLE X_pos,Y_pos,Radius,Alpha,Beta

PEEK(Adresse) ..... 125

PELLIPSE X_pos,Y_pos,A,B ..... [Pe] 107
PELLIPSE X_pos,Y_pos,A,B,Alpha,Beta

PI ..... 91

PLOT X_pos,Y_pos ..... [Pl] 108

POINT X_pos,Y_pos ..... 109

POKE Adresse,Byte ..... [Po] 126

POLYFILL Pkte,Xp(),Yp() ..... [Polyf] 110
POLYFILL Pkte,Xp(),Yp() OFFSET Xdiff,Ydiff

POLYLINE Pkte,Xp(),Yp() ..... [Pol] 109
POLYLINE Pkte,Xp(),Yp() OFFSET Xdiff,Ydiff

POLYMARK Pkte,Xp(),Yp() ..... [Polym] 110
POLYMARK Pkte,Xp(),Yp() OFFSET Xdiff,Ydiff

POS(Dummy) ..... 26

PRBOX X_links,Y_oben,X_rechts,Y_unten ..... [Prb] 111

PRINT AT(S,Z)[;,']Ausdruck[;,']V_name ..... [P bzw. ?] 26

PRINT #Kanalnummer,"Text"[;,']V_name[;,']V_name$[;,'] ..... 50

PRINT USING "format",Ausdruck$,V_name$,Ausdruck,Vname [P Using] 27

PRINT #Kanalnummer, USING "format",Liste ..... 51

PROCEDURE Name(Lv_name%,Lv_name,Lv_name$) ..... [Pro...Ret] 70
RETURN

PSAVE "Programmname" ..... [Ps] 39

PTSIN ..... 160

PTSOUT ..... 160

PUT x,y,V_name$,modus ..... [Pu] 114

PUT #Kanalnummer,Satznummer ..... [Pu] 51

QUIT ..... [Q] 71

RANDOM(n) ..... 92

RBOX X_links,Y_oben,X_rechts,Y_unten ..... [Rb] 111

READ V_name ..... [Rea] 71

RELSEEK #Kanalnummer,Byteanzahl ..... [Rel] 41

REM Kommentar ..... [R bzw. '] 72

REPEAT ..Befehle.. UNTIL ..... [Rep..U] 52

RESERVE Anzahl ..... [Rese] 179

RESTORE Label ..... [Res] 72

RESUME ..... Resu 62
RESUME NEXT
RESUME Label

RIGHT$(Zielstring,Anzahl) ..... 78

RMDIR "Ordnername" ..... [Rm] 39

RND(Dummy) ..... 92

RSET Ziel_var$=Stringausdruck ..... [Rs] 79

RUN ..... [Ru] 73

SAVE "Ordnername" ..... [Sa] 39

SDPOKE gerade Adresse,Word ..... [Sd] 127

SEEK #Kanalnummer,Bytenummer ..... [See] 42

SETCOLOR Register,Rotanteil,Grünanteil,Blauanteil ..... [Se] 104
SETCOLOR Register,Mischwert

SETTIME Zeit$,Datum$ ..... [Sett] 160

SIN(x) ..... 93

SGET Var$ ..... [Sg] 180

SGN(x) ..... 92

SHOWM ..... [Sh] 181

SLPOKE gerade Adresse,Long ..... [Sl] 127

SOUND Kanal,Volume,Note,Oktave,Dauer ..... [So] 53
SOUND Kanal,Volume,#Periode,Dauer

SPACE$(Anzahl) ..... 79

SPC(Anzahl) ..... 29

SPOKE Adresse,Byte ..... [Sp] 127

SPRITE Defstring$,x,y ..... [Spr] 111

SPUT Var$ ..... [Spu] 181

SQR(x) ..... 93

STOP ..... [St] 73

STR$(Wert) ..... 120

STRING$(Anzahl,Stringausdruck) ..... 80
STRING$(Anzahl,ASCII)

SUB V_name1,V_name2 ..... [S] 86
SUB V_name1,const

SWAP V_name1,V_name2 ..... [Sw] 86

SYSTEM ..... [Sy] 73

TAB(Position) ..... 93

TAN(x) ..... 93

TEXT x,y,"Text" ..... [T] 112
TEXT x,y,Stringausdruck
TEXT x,y,V_name$
TEXT x,y,Länge,V_name$

TIME$ ..... 161

TIMER ..... 74

TITLEW Handle,Stringausdruck ..... [Tit] 141

TROFF ..... 63

TRON #Kanalnummer ..... 63

TRUE ..... 181

TRUNC(x) ..... 94

TYPE(Zeiger) ..... 182

UPPER$(Stringausdruck) ..... 80

VAL(Stringausdruck) ..... 121

VAL?(Stringausdruck) ..... 122

VARPTR(V_name) ..... 133

VDIBASE ..... 182

VDISYS Opcode ..... 153

VOID Funktion(Parameter) ..... 184

VSYNC ..... 184

WAVE Kanal,Hüllkurve,Kurvenform,Periode,Dauer ..... [Wa] 54

WHILE ..Befehle.. WEND ..... [W..We] 58

dpeek (WINDTAB) ..... 142
dpoke WINDTAB,Wert

WRITE "Text","Text",V_name,V_name$,Ausdruck; ..... [Wr] 30
52

WRITE #Kanalnummer,"Text",V_name,V_name$ ..... 52

XBIOS(Opcode,Parameterliste) ..... 153

## 15.5 Stichwortverzeichnis


== ..... 162

Abbruch einer Schleife ..... 59
Abfrage einer Bedingung ..... 60
Abfrage eines Ports ..... 46
Abfrage von CTRL/SHIFT/ALT ..... 61
Abfrage von Fehlern ..... 61
Abgerundetes Rechteck ..... 111
Abspeichern eines Programms ..... 39
Absolutwert ..... 87
Abwarten des Vertikal-Blank ..... 184
Addition ..... 81, 85
Addrout-Array ..... 154
Adresse der Variablen ermitteln ..... 133
Adresse des Felddescriptors ..... 133
Ändern des Filepointers ..... 41
Äquivalenz ..... 83
AES ..... 186
AES - Adreß-Inputblock ..... 154
AES - Adreß-Outputblock ..... 154
AES - Anfangsadresse ..... 157
AES - Integer-Inputblock ..... 158
AES - Integer-Outputblock ..... 159
AES - Kontrollblock ..... 158
AES abschalten ..... 207
AES-Kontrolle ..... 188
AES-Routinen aufrufen ..... 151
Aktuelles Laufwerk ..... 34
Aktuelles Laufwerk bestimmen ..... 196
Aktuelles Laufwerk ermitteln ..... 197
Alertbox erstellen ..... 134
Alphanumerische Zeichen ..... 75
Anfang einer Prozedur ..... 70
Anfügen eines Kommentar ..... 165
Append ..... 48

Applikationshandle ermitteln ..... 186
Arcustangens ..... 88
Arithmetikbefehle ..... 81
Art der Linen ..... 98
ASCII-Code ..... 22
ASCII-Listing ..... 37
ASCII-Zeichen ..... 116f
Assemblierte Maschinenprogramme aufrufen ..... 128
Atari-BASIC-Format ..... 118f
Attribute der Menüpunkte ..... 146
Auflisten eines Programmes ..... 37
Aufruf einer Funktion ..... 68
Aufruf eines Programms ..... 73
Aufruf von Prozeduren ..... 59, 62
Aufruf von TOS-Funktionen ..... 153
Aufruf von VDI-Routinen ..... 152
Aufrufen des Maschinen-Monitor ..... 175
Ausdrucken des Programms ..... 43
Ausgabe an Peripherieport ..... 50
Ausgabe des Directory ..... 34, 36
Ausgabe des Ordnernames ..... 35
Ausgabe formatierter Daten ..... 27, 51
Ausgabe von Daten ..... 26, 30
Ausgabe von Text ..... 112
Ausgabe von Tönen ..... 53
Auslesen des Datenkanals ..... 47
Ausschalten des Mauszeigers ..... 173
Auswählen einer Datei ..... 135

BASIC-Basepage ermitteln ..... 166
BASIC-Bereich ..... 174
BASIC-Speichergröße bestimmen ..... 179
Bedingte Schleife ..... 57
Bedingungsabfrage ..... 60
Beenden des Interpreters ..... 71
Beenden einer Schleife ..... 59
Befehle an Compiler übergeben ..... 178
Beginn einer Prozedur ..... 70
Behandlung von Fehlern ..... 61
Beispielprogramme ..... 22, 97, 100, 130, 139
..... 164, 165, 172, 183
Bemerkungen anfügen ..... 72
Bereich in Teildatei speichern ..... 169
Bereich laden ..... 32
Beschreiben des Speichers ..... 126
Bestimmen der Linienfarbe ..... 95
Bestimmen der Mausform ..... 99
Bestimmen der Polygon-Eckpunkte ..... 110
Bestimmen des aktuellen Laufwerk ..... 196
Bestimmen des Feld-Startindex ..... 178
Bestimmen des Grafik-Text-Modus ..... 101
Bestimmen des Menüpunkt-Attribute ..... 146
Betrag einer Zahl ..... 87
Bewegen eines Speicherbereichs ..... 169
Biene ..... 99
Bildschirm löschen ..... 64
Bildschirm setzen ..... 181
Bildschirm speichern ..... 180
Bildschirmauflösung ermitteln ..... 202
Bildschirmausdruck ..... 43
Bildschirmbereich setzen ..... 114
Bildschirmbereich speichern ..... 114
Bildschirmoperationen ..... 26, 114
Bildschirmposition ..... 27
Binärwert ..... 117
BIOS ..... 153, 199
BIOS-Parameterblock-Adresse ..... 200
Boolsche Variablen ..... 129

Box im Rahmen verschieben ..... 190
Box-Rahmen ..... 194
Buchstabenumwandlung ..... 80

Charakter-Wert ..... 117
Compiler-Befehle übergeben ..... 178
CONTRL-Array ..... 154
Cosinus ..... 88
Cursor positionieren ..... 27
Cursorspalte ..... 170
Cursorspalte ermitteln ..... 26
Cursorzeile ..... 170

DATA-Zeiger setzen ..... 72
Datei auswählen ..... 135
Datei löschen ..... 37
Datei suchen ..... 36, 198
Datei umbennen ..... 38
Dateiauflistung ..... 36
Dateiende ermitteln ..... 40
Dateihandhabung ..... 40
Dateilänge ermitteln ..... 41
Dateiname ..... 49
Daten an Drucker ..... 44
Daten an Port ..... 50
Daten ausgeben ..... 26
Daten formatieren ..... 51
Daten in Kanal schreiben ..... 50
Daten sequentiell schreiben ..... 52
Datenausgabe ..... 30
Dateneingabe ..... 23
Datenfelder ..... 45
Datenkanal einrichten ..... 48
Datenkanal, Lesezugriff ..... 47
Datenkanal schließen ..... 45
Datensatz lesen ..... 46
Datensatz schreiben ..... 51
Datensatz unterteilen ..... 45
Datenspeicher ..... 65

Datentyp ..... 23
Datenumwandlung ..... 116
Datenzuweisung ..... 69
Datum einstellen ..... 160
Debugging einschalten ..... 63
Definieren einer Funktion ..... 66
Definition der Füllmuster ..... 96
Definition der Markierungssymbole ..... 99
Definition der Mausform ..... 99
Definition einer Prozedur ..... 70
Dekadischer Logarithmus ..... 90
Dekrementieren ..... 85
Desktop ..... 71
Dezimal-Zufallszahl ..... 92
Dezimalstellen-Funktion ..... 89
Dialogabwicklung ..... 134
Dicke der Linien ..... 98
DIGITAL RESEARCH ..... 185
Dimensionieren eines Feldes ..... 123
Directory ausgeben ..... 34
Directory erweitertes ..... 36
Directoryeintrag suchen ..... 36
Direktmodus ..... 65, 163, 208
Disjunktion ..... 83
Diskbuffer-Adresse ..... 197
DIsketteninhalt ..... 36
Diskettenoperationen ..... 31
Diskettenspeicherplatz ..... 34
Diskettenwechsel feststellen ..... 200
Disksektoren schreiben/lesen ..... 199
Division ..... 81, 85
Dreiecksignal ..... 55
Drive, aktuelles ..... 34
Drucken des Programmlistings ..... 43
Drucken, Zeichenkette mehrmals ..... 80
Druckeranweisungen ..... 43
Druckerattribute lesen/setzen ..... 206
Druckkopfposition ermitteln ..... 44
Dummy ..... 26

Eckpunkte eines Polygons ..... 110
Editieren eines Strings ..... 173
Editor ..... 67, 209
Editormenü ..... 37
Ein-/Ausgabebefehle ..... 21
Eingabe einer Zeichenkette ..... 24, 25
Eingabe von Tastatur ..... 23
Eingabekommentar ..... 25
Eingabestring ..... 24
Einlesen einer Zeichenkette ..... 48
Einlesen eines Datensatzes ..... 46
Einrichten eines Datenkanals ..... 48
Einschalten des Mauszeiger ..... 181
Einsetzen, String rechtsbündig ..... 79
Einsetzen, Zeichenkette linksbündig ..... 77
Einstellen der Farbregister ..... 104
Einstellen der Uhrzeit ..... 160
Einstellen des Datums ..... 160
Einstellen des Soundregisters ..... 54
Einzelschrittmodus ..... 63
Einzelvariablen löschen ..... 169
Ellipse zeichnen ..... 107
Ende der Datei ..... 40
Ende des Programms ..... 68
Ende einer Prozedur ..... 73
Ende einer Schleife ..... 59
Endlosschleife ..... 56
Ereignisfeststellung ..... 148
Ereignispuffer ..... 150
Ereignispuffer lesen ..... 186
Ereignispuffer schreiben ..... 187
Ereugen eines Ordners ..... 38
Erhöhen einer Variable ..... 85
Ermitteln der Druckkopfposition ..... 44
Ermitteln der Fehlerart ..... 156
Ermitteln der System-Uhrzeit ..... 161
Ermitteln der VDI-Base ..... 182
Ermitteln des aktuellen Laufwerks ..... 197

Ermitteln des Dateiendes ..... 40
Ermitteln des Fehlercode ..... 156
Ermitteln des Port-Ausgabestatus ..... 179
Ermitteln des Port-Eingabestatus ..... 174
Ermitteln des Speicherplatz ..... 157
Ermitteln des Systemdatums ..... 155
Ermitteln des Variablen-Typs ..... 182
Erniedrigen ..... 85
Ersetzen eines Teilstring ..... 175
Erstellen der Menüzeile ..... 147
Erstellen einer Alertbox ..... 134
Erweitertes Directory ..... 36
EVENT-Puffer ..... 144
Exklusiv-Oder-Verknüpfung ..... 83
Exponentialfunktion ..... 88

Fadenkreuz ..... 99
Farbbetrieb ..... 95
Farbe ..... 95
Farbmischung ..... 195
Farbregister lesen/setzen ..... 104, 203
Farbwert für Punkt ermitteln ..... 109
Fehler simulieren ..... 156
Fehlerart ermitteln ..... 156
Fehlerbehandlung ..... 62
Fehlerliste ..... 219
Fehlerverzweigung ..... 61
Feld dimensionieren ..... 123
Feld initialisieren ..... 123
Feld löschen ..... 64, 124
Feld-Startindex bestimmen ..... 178
Felddescriptor-Adresse ermitteln ..... 133
Feldelemente-Anzahl ..... 124
Feldröße ..... 45
Fenster auf Bildschirmgröße ..... 137
Fenster schließen ..... 137
Fenster öffnen ..... 138
Fenster-Informationszeile ..... 138
Fenster-Titelzeile ..... 141

Fenster-Verwaltungstabelle ..... 142
Fensterinhalt löschen ..... 137
Festlegen des Filepointers ..... 42
Festlegen einer Funktion ..... 66
Feststellen der Druckkopfposition ..... 44
File auswählen ..... 135
Filepointer ..... 40
Filepointer setzen ..... 42
Filepointer verschieben ..... 41
Filepointer-Position ..... 40
Fläche mit Muster füllen ..... 108
Format des Listings ..... 67
Formatieren von Tracks ..... 205
Formatierte Ausgabe ..... 27, 51
Formatierte Eingabe ..... 21
Formatsymbole ..... 28
Formatzahlen ..... 118
Formatzeichen ..... 26
Formular zentrieren ..... 188
Fortsetzen des Programms ..... 62
Funktion definieren ..... 66
Funktionen, numerische ..... 87
Funktionsaufruf ..... 68
Funktionsaufruf ohne Rückgabe ..... 183
Füllmuster definieren ..... 96

Ganzzahl-Funktion ..... 89, 94
GCONTRL-Array ..... 158
Gefülltes Polygon ..... 110
GEM-Window ..... 138
GEMDOS ..... 153
GEMDOS-Aufrufe ..... 196
Gerade Zahl testen ..... 171
GfA-BASIC V2.0 ..... 163
GfA-BASIC-Compiler ..... 178
GfA-BASIC-Format ..... 118
GINTIN-Array ..... 158
GINTOUT-Array ..... 159
Glücksspiele ..... 92

Grafik ..... 95
Grafik-Text-Modus bestimmen ..... 101
Grafikattribute ..... 195
Grafikbefehle ..... 105
Grafikbereich definieren ..... 194
Grafikdefinition ..... 95
Grafikmodus bestimmen ..... 103
Grafikparameter ..... 95
Grafiktext ..... 112
Großschreibung ..... 80
Größter Wert einer Liste ..... 90

Hardcopy ..... 43
Hexadezimal-Wert ..... 119
Hüllkurve ..... 54

Implikation ..... 83
Indentifikatoren ..... 47
Inhaltsverzeichnis einer Diskette ..... 36
Initialisieren eines Feldes ..... 123
Inkremetieren ..... 85
Input ..... 48
Integer-Zufallsfunktion ..... 92
Integerwerte ..... 89
Intergervariablen ..... 57
Interpreter verlassen ..... 71ff
INTIN-Array ..... 159
INTOUT-Array ..... 159
Invertieren des Menütitels ..... 148

Kanal auslesen ..... 47
Kanal beschreiben ..... 50
Kanalnummer ..... 49
Kennzeichnen des Variablenpointers ..... 166
Klangausgabe ..... 53f
Kleinschreibung ..... 80
Kompilierte Programme aufrufen ..... 127
Kommentar anfügen ..... 165
Kommentare ..... 72

Konjunktion ..... 82
Kontrollbefehle ..... 59
Konventionen ..... 10
Kopf einer Prozedur ..... 70
Kreis zeichnen ..... 106
Kreiszahl ..... 91

Label ..... 60
Laden einer Teildatei ..... 168
Laden eines Programmes ..... 37
Laden eines Speicherbereichs ..... 32
Laufwerk, aktuelles ..... 34
Laufwerk bestimmen ..... 196
Laufwerke, angeschlossene ermitteln ..... 201
Laufwerksbezeichnung ..... 31
Laufzeit ermitteln ..... 74
Leerzeichen ..... 79
Leerzeichen ausgeben ..... 29
Lesen der Farbregister ..... 203
Lesen des Speicherinhalts ..... 125
Lesen eines Datenkanals ..... 47
Lesen eines Datensatzes ..... 46
Lesen eines Ports ..... 46
Lesen von Disksektoren ..... 199
Lesen von Druckerattributen ..... 206
Lesen von Sektoren ..... 204
Linie zeichnen ..... 108
Liniendicke ..... 98
Linienfarbe bestimmen ..... 95
Linienmodi ..... 98
Linksbündige Zeichen in String ..... 76
Listen eines Programmes ..... 37
Listingformat ..... 67
Listschutz ..... 39
LOAD ..... 37
Logarithmus-Funktion ..... 90
Logische Bildschirmadresse ..... 202
Logische Operatoren ..... 82
Lokale Variablen ..... 69

Länge einer Datei ..... 41
Länge eines Strings ..... 77
Löschen der Menüzeile ..... 148
Löschen des Bildschirms ..... 64
Löschen des Programmspeichers ..... 70
Löschen einer Datei ..... 37
Löschen einer Einzelvariablen ..... 169
Löschen eines Fensters ..... 137
Löschen eines Ordners ..... 39
Löschen von Feldern ..... 64, 124
Löschen von Variablen ..... 64

Malen ..... 107
Markierungssymbole definieren ..... 99
Maschinen-Monitor ..... 175
Maschinenprogramme aufrufen ..... 127f
Maskenfarbe ..... 100
Math. Funktionen definieren ..... 66
Math. Operationen ..... 81
Matrix ..... 123
Mausereignis ..... 149, 176
Mausform bestimmen ..... 99
Mausstatus ..... 136
Mauszeiger ausschalten ..... 173
Mauszeiger einschalten ..... 181
Maximalwert ..... 90
MBASIC-Format ..... 118ff
Menge der Feldelemente ..... 124
Menü-Programmierung ..... 144
Menüpunkt-Attribute bestimmen ..... 146
Menüpunkte ..... 147
Menütitel ..... 147
Menütitel invertieren ..... 148
Menüzeile erstellen ..... 147
Menüzeile löschen ..... 148
MIDI ansteuern ..... 206
Minimalwert ..... 91
Mitteilungs-Puffer ..... 150
Mittelteil eines Strings ..... 78

Modulo-Berechnung ..... 81
Modus der Linen ..... 98
Movebox aufrufen ..... 191
MPB-Adresse ermitteln ..... 199
Multi-Ereignis ..... 150
Multiplikation ..... 81, 86
Muster definieren ..... 96
Muster in Fläche füllen ..... 108

Nachkommastellen ..... 89
Nachladen ..... 33
Name einer Funktion ..... 66
Natürlicher Logarithmus ..... 90
Negation ..... 83
Notenausgabe ..... 53
Numerische Funktionen ..... 87

Objekt-Adresse ermitteln ..... 187
Objektbaum zeichen ..... 188
Objektkoordinaten ermitteln ..... 189
Oder-Verknüpfung ..... 83
Öffnen eines Datenkanals ..... 48
Öffnen eines Fensters ..... 138
Oktalwert ..... 120
Oktave ..... 53
Online ..... 43
Operatoren, arithmetische ..... 81
Operatoren, logische ..... 82
Ordner ..... 31
Ordner erzeugen ..... 38
Ordner löschen ..... 39
Ordner wechseln ..... 33
Ordnername ausgeben ..... 35
Output ..... 48

Parameterliste ..... 59
Pause ..... 70
Periode ..... 54
Peripherieanschluß abfragen ..... 46
Peripherieanweisungen ..... 45
Pfadstruktur ..... 31
Pfeil ..... 99
Phys. Bildschirmadresse ..... 202
PI ..... 91
Platzhalter ..... 31
Polygon zeichnen ..... 109f
Polygon-Eckpunkte ..... 110
Port-Ausgabestatus ermitteln ..... 179
Port-Eingabestatus ermitteln ..... 174
Portabfrage ..... 46
Portausgabe ..... 50
Position der Maus ..... 136
Position des Filepointers ..... 40
Positionierung des Cursors ..... 27
Potenzieren ..... 81
Prioritäten ..... 84
Privilegierter Modus ..... 127
Programm ausdrucken ..... 43
Programm ausführen ..... 171
Programm fortsetzen ..... 62
Programm laden ..... 37
Programm listen ..... 37
Programm speichern ..... 39
Programm starten ..... 73
Programm unterbrechen ..... 73
Programmabbruch ..... 65ff, 71
Programme verketten ..... 33
Programmlisting ..... 67
Programmschleife ..... 57
Programmschritte einzeln ausführen ..... 63
Programmschutz ..... 39
Programmspeicher löschen ..... 70
Programmsprung zu Label ..... 60
Programmstrukturen ..... 56

Programmtabellen ..... 65
Programmüberwachung ..... 63
Prozedur aufrufen ..... 59, 62
Prozeduranfang ..... 70
Prozeduren ..... 59
Prozedurende ..... 73
PTSIN-Array ..... 160
PTSOUT-Array ..... 160
Pulldown-Menütext ändern ..... 189
Punkte zeichnen ..... 107f
Punktfarbe ..... 109

Quadratwurzel ..... 93

RAM-Operationen ..... 125
Random ..... 49
Random-Zahl ..... 92
Rasterzeile ..... 184
Rauschen ..... 54
Rechteck, abgerundet ..... 111
Rechteck ausdehnen ..... 191
Rechteck schrumpfen ..... 192
Rechteck zeichnen ..... 105
Rechtsbündiger Stringteil ..... 78
RSC-File laden ..... 187
RSC-Speicher freigeben ..... 187
Rubberbox aufrufen ..... 190
Run-Only-Interpreter ..... 164
Runden von Werten ..... 170

Satznummer ..... 46, 51
Scan-Code ..... 22
Schleife, bedingt ..... 57
Schleife mit Zählvariable ..... 57
Schleifenabbruch ..... 59
Schließen des Datenkanals ..... 45
Schließt Fenster ..... 137
Schluß einer Prozedur ..... 73
Schreiben, auf Datenkanal ..... 50
Schreiben eines Datensatzes ..... 51
Schreiben mit Sektoren ..... 205
Schreiben von Daten (sequentiell) ..... 52
Schreiben von Disksektoren ..... 199
Schritte des Programms ..... 63
Screen löschen ..... 64
Screen-Parameter setzen ..... 203
Scrolling ..... 27
Sektor(en) lesen ..... 204
Sektor(en) schreiben ..... 205
Sektor(en) verifizieren ..... 206
Sequentielle Daten ..... 52
Setzen der Farbregister ..... 203
Setzen des Bildschirmbereichs ..... 114
Setzen des Bildschirms ..... 181
Setzen des Filepointers ..... 42
Setzen Druckerattribute ..... 206
Setzen eines Sprite ..... 111
Setzen eines Tabulators ..... 29
Signum-Funktion ..... 92
Simulieren eines Fehlers ..... 156
Sinus ..... 93
Sondertasten ..... 21
Sonderzeichen ..... 24, 75
Soundchip-Anweisungen ..... 53
Soundregister einstellen ..... 54
Spaces ..... 79
Spaces ausgeben ..... 29
Spalte der Cursorposition ..... 26
Speicher des Bildschirm ..... 180

Speicher freigeben ..... 198
Speicher reservieren ..... 198
Speicherbereich bewegen ..... 169
Speicherbereich laden ..... 32
Speicherbereiche verknüpfen ..... 167
Speicherinhalt lesen ..... 125
Speicherinhalt verändern ..... 126
Speichern eines Bereichs in Teildatei ..... 169
Speichern eines Bildschirmbereichs ..... 114
Speichern eines Programms ..... 39
Speicheroperationen ..... 125
Speicherplatz ..... 157
Sprite setzen ..... 111
Sprung zu Label ..... 60
Start einer Prozedur ..... 70
Starten eines Programms ..... 73
Startindex ..... 27, 178
Status der Maus ..... 136
Steuerzeichen ..... 75
STOP-Abfrage ..... 61
String an MIDI ..... 206
String editieren ..... 173
String links teilen ..... 76
String linksbündig einsetzen ..... 77
String rechtsbündig einsetzten ..... 79
Stringeingabe ..... 21
Stringkette ..... 80
Stringlänge ..... 77
Stringmitte ..... 78
Stringteil, rechts ..... 78
Stringumwandlung ..... 120
Stringwert ..... 121
Strukturbefehle ..... 64
Subtraktion ..... 81, 86
Suchen einer Datei ..... 36, 198
Suchen, Zeichenkette ..... 76
Suchpfad ..... 31
Summieren ..... 85
Supervisor-Poke ..... 127

Switch-Tastenstatus ermitteln ..... 192, 201
System-Uhrzeit ermitteln ..... 161
Systemadressen ..... 154
Systemaufrufe ..... 151
Systemdaten ..... 154
Systemdatum ermitteln ..... 155
Systemroutinen ..... 185
Sägezahnsignal ..... 55

Tabelle für Fensterverwaltung ..... 142
Tabellen ..... 65
Tabulator ..... 27ff
Tangens-Funktion ..... 93
Tastatur abfragen ..... 21
Tastatur-Ereignis ..... 150
Tastatur-Repeat einstellen ..... 207
Tastaturanweisungen ..... 21
Tastaturcodes ..... 163
Tastendruck ..... 21
Tastendruck abwarten ..... 47
Tauschen von Variablenwerten ..... 86
Teildatei laden ..... 168
Teilstring ersetzen ..... 175
Teilstring suchen ..... 76
Test auf gerade Zahl ..... 171
Test auf ungerade Zahl ..... 176
Test des Programms ..... 63
Text in Grafik ..... 112
Textart setzen ..... 193
Textgröße setzen ..... 193
Textoperationen ..... 75
Timer ..... 74
Titelzeile eines Fensters ..... 141
Tonausgabe ..... 53
Tonhöhe ..... 54
Tonregister einstellen ..... 54
TOS-Programm ausführen ..... 171

TOS-Routinen aufrufen ..... 153
TRACE-Modus ..... 63
Track formatieren ..... 205
Typ der Linien ..... 98

Überwachung des Programmablaufs ..... 63
Uhrzeit einstellen ..... 160
Umbennen einer Datei ..... 38
Umwandlung von Buchstaben ..... 80
Und-Verknüpfung ..... 82
Ungerade Zahl testen ..... 176
Unterbrechen eines Programms ..... 73
Unterbrechung ..... 61
Unterteilen eines Datensatzes ..... 45
Unterverzeichnis erzeugen ..... 38
Unterverzeichnis löschen ..... 39
Unterverzeichnis wechseln ..... 33
Unwahr-Konstante ..... 173
Update ..... 48

V_name ..... 23
Value ..... 121
Variable multiplizieren ..... 86
Variablen, lokale ..... 69
Variablen löschen ..... 64
Variablen subtrahieren ..... 86
Variablenadresse ermitteln ..... 133
Variablenorganisation ..... 129
Variablenpointer kennzeichnen ..... 166
Variablentyp ..... 129, 182
Variablenumwandlung ..... 120
Variablenwerte vertauschen ..... 86
Variablenzuweisung ..... 69
VDI ..... 193
VDI - Integer-Inputblock ..... 159
VDI - Integer-Outputblock ..... 159
VDI - Kontrollblock ..... 154
VDI - Punkt-Inputblock ..... 160
VDI - Punkt-Outputblock ..... 160

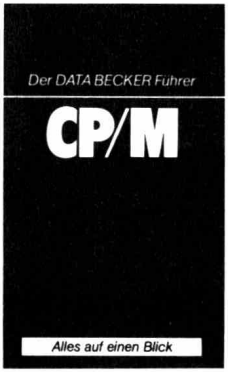
VDI-Feldbasis ermitteln ..... 182
VDI-Routinen aufrufen ..... 152
Vergleichsoperatoren ..... 82
Vergrößern des Fensters ..... 137
Verketten von Programmen ..... 33
Verlassen des Interpreters ..... 71, 73
Vermindern einer Variable ..... 85
Verschieben des Filepointers ..... 41
Version V2.0 ..... 162
Vertauschen von Variablenwerten ..... 86
Vertikal-Blank abwarten ..... 184
Verwaltungstabelle der Fenster ..... 142
Verzeigung zu Label ..... 60
Verzweigung bei Fehlern ..... 61
Verzweigung bei Menüauswahl ..... 149
Verzweigung zu Prozedur ..... 59, 62
Verzögerung ..... 70
Verändern des Speicherinhalts ..... 126
Volume ..... 53
Vorzeichen ..... 92

Wahr-Konstante ..... 181
Wartefunktion ..... 70
Wechseln des Ordners ..... 33
Wellenform ..... 54
Werte runden ..... 170
Werte tauschen ..... 86
Wertedefinition ..... 69
Wildcards ..... 31
Window öffnen ..... 138
Window-Infozeile ..... 138
Window-Programmierung ..... 137
Window-Titel ..... 141
Windtab-Tabelle ..... 142
Wurzel, quadratische ..... 93

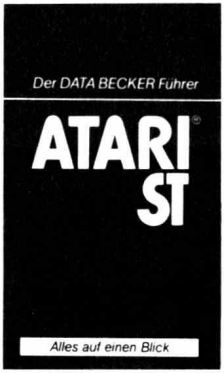
XBIOS ..... 153, 202

Zahl der Feldelemente ..... 124
Zahl der Ziffern ..... 122
Zeichenkette ..... 75
Zeichenkette aus Leerzeichen ..... 79
Zeichenkette eingeben ..... 24f
Zeichenkette einlesen ..... 48
Zeichenkette einsetzen ..... 79
Zeichenkette in einem String suchen ..... 76
Zeichenkette linksbündig einsetzen ..... 77
Zeichenkette mehrmals drucken ..... 80
Zeichenumwandlung ..... 80
Zeichnen einer Ellipse ..... 107
Zeichnen einer Linie ..... 108
Zeichnen eines Kreises ..... 106
Zeichnen eines Polygon ..... 109
Zeichnen eines Punktes ..... 107f
Zeichnen eines Rechtecks ..... 105
Zeigeroperationen ..... 129
Zeilenscrolling ..... 27
Zeilenüberlauf ..... 27
Zeitabfrage ..... 74
Zufallsfunktion ..... 92
Zugriffspfad ..... 31
Zählschleife ..... 56

In der Regel klappt sie phantastisch, die Arbeit mit dem Computer. Und für Zweifelsfälle hat man ja bereits eine ansehnliche Bibliothek nützlicher Literatur. Doch immer wieder – mitten in der Arbeit – passiert es: Man sucht nach einem bestimmten Kommando. Irgendwo im Handbuch, oder stand es in einem Computermagazin... Der Arbeitsfluß ist unterbrochen. Man versucht sich zu erinnern, durchwühlt den riesigen Literaturberg, sucht einen Hinweis. HILFE. Genau die bekommen Sie von den neuen DATA BECKER Führern. Ein gezielter Griff und Sie haben die gewünschte Information. Hier finden sie umfassend alles auf einem Blick. Zu Ihrem Rechner oder auch zur entsprechenden Software. Das sind die ersten DATA BECKER Führer:




**Der DATA BECKER Führer**
**zu CP/M**
**139 Seiten**
**DM 19,80**




**Der DATA BECKER Führer**
**zum ATARI ST**
**ca. 200 Seiten**
**ersch. ca. Anfang Dez.**
**DM 29,80**

Wie gesagt, dies ist erst der Anfang. Weitere DATA BECKER Führer werden folgen: First Word/First Lektor, C16, CPC, WordStar, MS-DOS und, und, und. Immer aktuell, übersichtlich und hilfreich. Immer im gleichen Gewand: Robustes Hardcover im handlichen Westentaschenformat. Und immer gleich strukturiert: Alle Befehle und Funktionen nach Sachgruppen, alphabetisch mit Kurzsyntax und über Stichworte geordnet. Egal, wie sich Ihr Problem darstellt, mit einem Blick in den DATA BECKER Führer ist es bereits gelöst.

ISBN 3-89011-402-4

DM 24,80
SFr 24
ÖS 193